Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRAdO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Aplicación móvil para potenciar la creación de relaciones profesionales previas a eventos o convenciones

Estudiante: Gabriel Castro Penedo
Dirección 1: Óscar Fresnedo Arias
Dirección 2: Óscar Naverias Picos

A Coruña, setembro de 2020.

A mi familia, amigos y mentores

Agradecimientos

Agradecimientos a mi familia y amigos por apoyarme durante el transcurso de mis estudios, a Javier Freire por introducirme en el mundo del desarrollo, a Óscar Naveiras y Javier Sanesteban por brindarme la oportunidad de formarme y realizar este proyecto, a todo el equipo de Kelea y a Óscar Fresnedo como tutor por su ayuda durante todo el desarrollo.

Resumen

El objetivo de este trabajo de fin de grado es desarrollar una aplicación móvil para iOS y Android que ayude a los participantes en eventos a aumentar el número de relaciones que realizan durante el transcurso de estos. Para ello, se implementará una aplicación en Flutter que sea capaz de agrupar a estos participantes por intereses y realizar videollamadas entre ellos, para que si se valoran positivamente se comparta su información de contacto.

Abstract

The objective of this final degree project is to develop a mobile application for iOS and Android that helps event atendees to increase the number of relationships they make during its course. To complete this goal, the application will be implemented in Flutter and will be capable of grouping these atendees by interests and making video calls between them, so that if they are positively valued, their contact information is shared.

Palabras clave:

- Flutter
- Scrum
- Algoritmo genético
- Contacto
- Reunión
- Intereses
- Videollamadas

Keywords:

- Flutter
- Scrum
- Genetic algorithm
- Networking
- Meeting
- Interests
- Video calls

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se introducirá el proyecto, hablaremos sobre el contexto del mismo y sobre la motivación que hizo que este se llevara a cabo. Además, se expondrán los objetivos principales que se intentarán alcanzar a lo largo de su desarrollo, así como la estructura en la que se dividirá esta memoria.

Actualmente, estamos en una época en la que la tecnología ha cambiado radicalmente la forma en la que nos relacionamos. El aumento constante de personas que utilizan las tecnologías móviles y la digitalización de servicios hace que sea muy común que las personas usen cada vez más dispositivos electrónicos en su día a día. En especial, uno de los cambios más llamativos es que conocer a nuevas personas o hacer contactos ya no es un acto totalmente presencial, sino que se convierte en un acto también digital. De la misma forma, toda la comunicación posterior se puede hacer de forma telemática sin necesidad de interacción física.

La situación excepcional motivada por el COVID ha aumentado drásticamente el uso de herramientas tecnológicas para la comunicación entre personas de manera remota, tanto en los ámbitos profesionales como en un contexto educativo o personal. Además, como consecuencia directa de este aumento, están surgiendo constantemente nuevas ideas para mejorar las herramientas que ya daban solución a estos problemas, al igual que nuevas aplicaciones que intentan abrirse hueco en este mercado aportando ideas innovadoras.

En el ámbito profesional, la situación ha obligado a realizar grandes cambios en la forma de trabajar de los empleados y en la forma en la que la empresa distribuye sus productos o servicios. Estos cambios producen un gran coste tanto a nivel económico como de esfuerzo por parte de todos los trabajadores, por lo que cualquier herramienta que ayude a mejorar la nueva forma de trabajar y comunicarse es posible que reciba una gran acogida [1].

1.1 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación móvil comercial, en este caso usando Flutter, que consiga maximizar el número de relaciones profesionales que se pueden establecer durante eventos o convenciones. Este aumento en el número de relaciones profesionales se consigue gracias a la realización de videollamadas previas a los eventos presenciales, en las que los usuarios conocerán a otros asistentes y valorarán las primeras impresiones que han tenido de ellos. La aplicación intentará, en la medida de lo posible, agrupar a los usuarios por gustos o intereses comunes para ayudar a que las videollamadas generen el máximo valor posible.

Aunque principalmente se centre en eventos, la aplicación podría usarse en más ámbitos, como en el caso de empresas grandes que buscan aumentar las relaciones entre trabajadores de distintos proyectos. Este proyecto estará centrado básicamente en el objetivo principal, pero siempre con la vista puesta en los posibles usos a mayores que se le podrá dar en un futuro y, por tanto, la modularidad de esta es esencial.

Las funcionalidades a destacar de la aplicación que se va a desarrollar son las siguientes:

- Puesta en contacto de personas con intereses similares a través de la realización de videollamadas cortas. Para ello, la aplicación debe permitir que un usuario se pueda registrar en sesiones virtuales creadas para un evento. Para cada sesión, se realizarán varias rondas de videollamadas en las que cada participante se reunirá con diferentes grupos de participantes que tengan intereses afines.
- Clasificación de las sesiones de videollamada en grupos de interés para aumentar el grado de coincidencia de los participantes.
- Compartición de los datos de contacto en caso de que los usuarios se evalúen positivamente de forma bilateral.
- Mantenimiento de una agenda de sesiones.
- Mantenimiento de una agenda con los contactos realizados.

La idea para este proyecto es implementar un mínimo producto viable que sea suficientemente complejo para realizar pruebas con usuarios reales y demostrar la viabilidad de la aplicación. Para ello, también será necesario implementar una pequeña aplicación de administración en la que los organizadores de los eventos o charlas puedan crear sesiones virtuales

de *networking* asociadas a estos eventos para realizar la ronda de reuniones por videollamada. Esta parte de administración será mucho más sencilla que la aplicación para los usuarios finales. Por ello, en este caso, se ha decidido implementarla en formato web y también se realizará en Flutter, aprovechando que este tiene la ventaja de que permite compilar tanto para móvil como para web.

1.2 Motivación

La idea de desarrollar este producto surgió para cubrir una necesidad real dentro de una empresa, lo que aumenta las posibilidades de que este problema también haya surgido en otras empresas y que sea un producto que encaje bien en el mercado. Esta circunstancia puede ayudar, en el momento de la salida al mercado de la aplicación final, a distribuirla entre otros organizadores de eventos a los que les pueda interesar tener estas sesiones de contacto previas al evento entre sus participantes, o para empresas que quieran potenciar sus relaciones internas.

Dado que el proyecto se realiza dentro de unas prácticas de empresa y que esta empresa se dedica, principalmente, a asesorar a otras empresas para aplicar metodologías modernas de desarrollo en sus proyectos, se aprovechará este conocimiento para realizar el desarrollo de la aplicación aplicando una de estas metodologías con el objetivo de ganar experiencia de cara al futuro profesional. Como la empresa también tiene experiencia en la organización y patrocinio de eventos, su *feedback* puede ser de gran ayuda para que la aplicación se adapte bien a las posibles necesidades que tengan los organizadores de eventos.

Como se ha comentado, la aplicación se va a desarrollar en Flutter, lo que permite implementar soluciones multiplataforma de una manera mucho más rápida y sencilla que utilizando diferentes lenguajes nativos. Esto hace que sea totalmente viable que una persona pueda desarrollar en un tiempo razonable una aplicación completa, con bastantes funcionalidades, como es el caso del proyecto que se va a realizar para este trabajo fin de grado. De esta forma, otra de las motivaciones que ha impulsado el desarrollo de este proyecto es comprobar la viabilidad de Flutter como plataforma para el desarrollo de aplicaciones finales y la velocidad con la que se pueden implementar productos usando este *framework*.

1.3 Estructura de la memoria

Durante el resto de este documento, se explicarán las diferentes fases por las que pasará el desarrollo de este proyecto. A continuación, se presenta un breve resumen de lo que incluirá

cada capítulo de la memoria:

- **Estado del arte:** Aplicaciones que cubren nichos de mercado similares a la aplicación del proyecto.
- **Fundamentos tecnológicos:** Explicación de las tecnologías que se usarán durante el ciclo de desarrollo.
- **Metodología:** Explicación sobre el método de trabajo con el que se desarrollará la aplicación.
- **Planificación y análisis económico:** Aproximación del coste monetario y temporal que tendría la aplicación en caso de que fuese desarrollada por un equipo completo.
- **Análisis de requisitos:** Análisis de los casos de uso y requisitos funcionales y no funcionales que se implementarán más tarde.
- **Diseño:** Explicación del diseño de interfaz que se ha decidido implementar.
- **Implementación:** Explicación del proceso de implementación.
- **Conclusiones.**

Capítulo 2

Estado del arte

En este capítulo se expondrán los distintos tipos de aplicaciones que abordan problemas similares a los que intenta dar solución este proyecto y, junto a ello, se destacarán aquellas ideas que son interesantes para ser incorporadas en la aplicación. Lo dividiremos en tres partes, una en la que se incluyen aplicaciones hechas para conocer gente, otra con aplicaciones pensadas para crear redes profesionales y, por último, una parte con aplicaciones que permiten comunicación mediante videollamadas.

2.1 Aplicaciones para conocer a otras personas

Uno de los objetivos de el proyecto, y posiblemente el más importante, es implementar una aplicación que permita hacer "*networking*", es decir, hacer contactos y establecer relaciones profesionales entre usuarios. Por ello, es interesante analizar aplicaciones que ya hayan tenido éxito haciendo que los usuarios conozcan a gente nueva. Normalmente este tipo de aplicaciones están enfocadas a que los usuarios busquen una pareja sentimental, y aunque algunas nacieron con otros propósitos, la mayoría suelen siempre derivar a este tipo de relaciones. Sin embargo, lo interesante no es a que están enfocadas, sino como han hecho que los usuarios se sientan cómodos utilizándolas y la forma en la que calculan quién podría ser tu pareja ideal.

2.1.1 Meetic

Meetic [2] es la primera aplicación que se dio a conocer, saliendo al mercado públicamente en 2005 y hoy en día sigue siendo una de las más usadas entre las personas que deciden buscar pareja a través de internet, pero su actividad no es sólo *online*, Meetic también organiza eventos para que solteros y solteras puedan conocerse presencialmente. Para buscar pareja a sus usuarios, la aplicación requiere que en el momento de registrarte contestes a una encuesta sobre tus gustos y sobre las restricciones que tu pareja ideal debe de cumplir. Desafortunada-

mente, y al igual que hacen la gran mayoría de aplicaciones de citas, esta no comparte la forma en la que funciona el algoritmo de "*matching*". Es posible que se usen funciones matemáticas complejas que analicen las respuestas a la encuesta que se cubre en el registro, o puede que simplemente busque personas de la misma edad en una zona cercana a la del usuario.

Meetic nos confirma que hay usuarios interesados en conocerse presencialmente en quedadas aunque estos se hayan registrado para tener citas con una sola persona a través de sus intereses y, desde el punto de vista del proyecto que se va a realizar, afirma que la idea de conocer gente previamente a un evento es algo que se ha probado y ha dado un buen resultado a largo plazo.

2.1.2 Tinder

Tinder [3] es otra de las aplicaciones más usadas actualmente para establecer relaciones personales, sobre todo entre los jóvenes. En 2014 ya contaba con más de 50 millones de usuarios registrados. La aplicación se basa en mostrarle al usuario personas cercanas dentro del rango de edad que ha seleccionado, y que este decida si le ha gustado mirando sus fotografías y una pequeña descripción. Nunca sabes si los otros usuarios te han votado positiva o negativamente pero, en el momento de que haya votos positivos entre dos personas, se hará el *match* y se compartirá el contacto.

Tinder aporta una idea muy interesante para este proyecto. Esta idea es que, aunque te hayas reunido con cientos de personas (en el caso de este proyecto mediante videollamadas), sólo se compartirá la información de contacto en caso de que haya una buena puntuación mutua entre los usuarios, y nunca sabrás qué puntuación te ha dado cada persona. Esto favorece que los usuarios voten con mayor sinceridad y que su voto no se vea influenciado por la opinión que tenga el otro usuario sobre el voto que ha recibido.

2.2 Aplicaciones para crear redes profesionales

Como se ha comentado, la aplicación que se va a desarrollar estará enfocada, en un primer momento, a usuarios dentro de su ámbito profesional. Por eso es interesante conocer cuáles son las mejores aplicaciones utilizadas para este perfil de usuarios y conocer las funcionalidades que las diferencian de otro tipo de aplicaciones menos demandadas.

2.2.1 Linkedin

La red social para profesionales por excelencia. Un perfil en Linkedin [4] no suele mostrar información personal, sino que se muestra información laboral y la formación del usuario. En

Linkedin comienzas por añadir a personas ya conocidas y, a partir de tus contactos e intereses, te sugiere nuevos contactos para crear una red profesional mucho más amplia. Además, las empresas pueden publicar ofertas de empleo que se mostrarán a candidatos aptos. Han surgido diferentes alternativas durante todo este el tiempo pero ninguna consigue alcanzarla en número de usuarios y Linkedin parece una red totalmente consolidada.

De esta red social podemos concluir que es importante, si lo que queremos es que una aplicación se centre en el ámbito profesional, que los usuarios puedan añadir información sobre sus estudios y su experiencia laboral, ya que como se ha demostrado, ayuda a hacer contactos de tus mismos intereses o de tu mismo nivel jerárquico dentro de una empresa.

2.2.2 Meetup

Meetup [5] no es una red social propiamente dicha, sino que se centra en organizar quedadas dentro de distintos grupos de interés. Es interesante analizarla para este proyecto porque estas quedadas propician la creación de relaciones tanto personales como también profesionales, ya que las personas que pertenecen a un mismo grupo y asisten a los mismos eventos, comparten intereses o trabajan en el mismo gremio. Los grupos de interés pueden facilitar que las personas que pertenecen a dichos grupos creen relaciones de valor.

2.3 Aplicaciones de comunicación mediante videollamadas

El catálogo de aplicaciones con esta funcionalidad es muy extenso, y en la situación actual el número de usuarios se ha multiplicado. Las más destacadas en este grupos son aplicaciones como Zoom, Microsoft Teams o Google Meet, todas ellas permiten realizar videollamadas entre varias personas, crear salas e invitar a participantes, pero Teams tiene un añadido que permite crear grupos y organizar varias sesiones de videollamadas para estos grupos. Esta funcionalidad de la aplicación de Microsoft es quizás la más interesante y que nos puede aportar ideas de cara al desarrollo de este proyecto, dividiendo a los usuarios en grupos para las llamadas.

2.4 Conclusiones

Analizando brevemente estas tres categorías de aplicaciones, sacamos ciertas conclusiones sobre algunas funcionalidades que sería interesante implementar en el proyecto. Estas funcionalidades han sido ampliamente utilizadas por los usuarios de estas aplicaciones y, por lo tanto, ya ha sido demostrada su viabilidad, lo que nos asegura que son funcionalidades que

el usuario posiblemente vaya a demandar o esté acostumbrado a utilizar. Las conclusiones obtenidas se podrían resumir en los siguientes puntos:

- Agrupar a los usuarios mediante algún algoritmo que compare sus intereses.
- No mostrar las puntuaciones que los usuarios se dan entre sí, simplemente agrupar a los que se hayan puntuado positivamente.
- Permitir que los usuarios vean la experiencia laboral y formación de sus contactos.
- Permitir que los participantes de las sesiones de videollamada se agrupen por grupos de interés.

La aplicación a desarrollar aportará un conjunto de todas estas funcionalidades mencionadas anteriormente, combinando funciones interesantes de cada una y uniéndolas para crear un único producto, añadiendo la posibilidad de realizar de forma no presencial actividades de *networking* que normalmente se realizan durante eventos o reuniones presenciales. La aplicación debería cubrir un nicho de mercado que actualmente parece que no está explotado.

Capítulo 3

Fundamentos tecnológicos

En este capítulo se exponen las principales tecnologías, herramientas y programas que se han utilizado para el desarrollo del proyecto, así como la razón por la que se ha decidido seleccionarlas.

3.1 Dart

Dart [6] es el lenguaje de programación que se ha elegido para implementar tanto la interfaz de usuario (gracias al *framework* Flutter), como el *backend* de la aplicación. Dart es un lenguaje orientado a objetos creado por Google y optimizado para el desarrollo de interfaces gráficas, pero también es suficientemente potente como para crear funciones para el *backend*. Su sintaxis es muy similar a la de otros lenguajes, sobre todo a la de Java.

Una de sus principales ventajas es la capacidad de que los programas implementados en este lenguaje puedan ser compilados para todo tipo de plataformas, como Android, iOS o web, y que esto se realice sin una pérdida de rendimiento futura pues esta compilación transforma el código Dart en código nativo de cada plataforma. Otra de las funciones interesantes que este lenguaje soporta, es el uso de los futuros y las cláusulas Async-Await, que nos permite crear funciones que devuelvan datos que conoceremos en un futuro, ejecutarlas asíncronamente o esperar a que estas realicen todas sus operaciones antes de avanzar a la siguiente función. Esta es una funcionalidad muy potente, sobre todo para aplicaciones que hacen consultas a otras fuentes de datos o realizan operaciones complejas con tiempos de ejecución largos.

A parte de todo esto, la principal razón por la que se ha elegido Dart para implementar la parte de la aplicación móvil es que Dart es el único lenguaje que permite usar el *framework* de Flutter, del cual hablaremos en la siguiente sección. Para el *backend*, posiblemente haya mejores opciones que Dart, pero ya que se ha decidido usarlo para la parte de la interfaz

gráfica, es interesante para ver su viabilidad.

3.2 Flutter

Flutter [7] es un *framework* que, al igual que Dart, está desarrollado por Google. Su principal atractivo es que, con un único código, usando Flutter podemos compilar aplicaciones para móvil, web y escritorio. A continuación se explican las funciones más importantes y que distinguen Flutter del resto de *toolkits* multiplataforma.

3.2.1 Hot reload

El Hot Reload o reinicio en caliente, es una de las funciones más útiles y que permiten que Flutter sea posiblemente el *framework* con el que se pueden diseñar interfaces gráficas de forma más rápida. A diferencia de otros *toolkits*, con Flutter podemos hacer cambios en el código (y por consiguiente en la interfaz), y ver en tiempo real los cambios que hemos realizado en la pantalla de nuestro dispositivo, ahorrando tiempo al no tener que volver a compilar toda la aplicación después de cada cambio.

3.2.2 Interfaces bonitas y amigables

Flutter incorpora de serie una gran cantidad de elementos gráficos prediseñados bajos las líneas de diseño de Material Design (utilizado en las aplicaciones de Google y una gran parte de aplicaciones Andorid) o Cupertino (utilizado por los desarrolladores de aplicaciones para iOS). Esto permite que, sin un conocimiento amplio sobre diseño de interfaces, se puedan realizar aplicaciones con cierto atractivo visual y facilidad de uso.

3.2.3 Rendimiento nativo

Con Flutter las aplicaciones se compilan a código nativo de cada plataforma, por lo que el rendimiento es igual de bueno que si la aplicación se hubiese programado directamente en el lenguaje nativo. Flutter también tiene en cuenta las peculiaridades de cada plataforma en cuanto a animaciones o navegación entre pantallas, y las aplicaciones se compilan con estas peculiaridades de forma transparente para el programador.

3.2.4 Librería de paquetes

Flutter ofrece una cantidad casi infinita de paquetes con determinadas utilidades creados por la comunidad o por los propios desarrolladores de Google. Podemos encontrar tanto paquetes visuales que no están incluidos en el conjunto de *widgets* que se incorporan de serie, como paquetes para integrar otras Application Programming Interface (API) y plataformas.

Cualquier usuario puede subir su paquete a la web y quien decida utilizarlo también tiene la posibilidad de modificarlo a su gusto. Esto permite que los desarrolladores compartan sus creaciones de elementos que han considerado necesarios para sus aplicaciones y que puede que otros necesiten. Esto también es beneficioso para los usuarios ya que evita cargar el *framework* con una cantidad de elementos y librerías innecesarias, pues el propio usuario descarga solo aquellas que crea conveniente.

3.3 Firebase

Firebase [8] será el encargado de alojar todo el *backend* de la aplicación. Es una plataforma ubicada en la nube, que funciona en conjunto con Google Cloud Platform y que permite a los desarrolladores alojar en la misma una gran cantidad de funcionalidades a un coste muy bajo, con la ventaja de poder gestionar todas estas desde una misma interfaz. Entre la gran cantidad de servicios que ofrece, que se muestran en la figura 3.1, los que se utilizarán para este proyecto son los que se exponen en los siguientes subapartados.

3.3.1 Cloud Firestore

Es una base de datos alojada en la nube, en formato Not Only Structured Query Language (NoSQL). Se utilizará para guardar toda la información que no necesita ser consultada en tiempo real, como usuarios y sus datos, eventos o sesiones. Una ventaja de su uso es que, al estar alojada en la nube, permite desplegar aplicaciones sin disponer de un servidor propio para alojar los datos.

3.3.2 Realtime Database

Realtime Database es otra base de datos disponible dentro de los servicios de Firebase. Es también una base de datos NoSQL pero, en este caso, está optimizada para que las aplicaciones puedan crear, actualizar o eliminar datos en tiempo real y que estos cambios se sincronicen con todos los usuarios que estén accediendo a ella. Esta base de datos se usará para guardar toda la información correspondiente a las reuniones, pues estas se realizan en tiempo real, y puede guardar datos sobre los usuarios que están conectados o cuáles han acabado ya una llamada.

3.3.3 Cloud Storage

Es simplemente un método de almacenamiento de archivos en la nube con la ventaja de que, una vez subido el archivo, se genera una Uniform Resource Locator (URL) y el archivo

se puede descargar a partir de esta. Se utilizará para guardar las imágenes de perfil de cada usuario.

3.3.4 Firebase Authentication

Es un gestor de usuarios multiplataforma, proporciona autenticación a las aplicaciones que lo usan, ofrece todo tipo de mecanismos de autenticación como, por ejemplo, mediante teléfono, email y contraseña, Google, Facebook, Twitter o Apple. Además, permite añadir verificaciones por correo electrónico o Short Message Service (SMS) para confirmar la identidad del usuario que se registra.

3.3.5 Cloud Functions

Este servicio se encarga de alojar las funciones del *backend* de la aplicación que se ejecutan en un entorno Node.js. Estas funciones sólo se ejecutan en caso de que ocurran determinados eventos y pueden ser útiles para realizar mantenimientos o actualizar datos de las bases de datos. En el caso de este proyecto, se utilizarán para mantener actualizada la base de datos en tiempo real, para realizar los cálculos de las salas en las reuniones, o para enviar notificaciones en determinados momentos. La gran ventaja es que alojando esta lógica en Firebase, no se realiza carga de trabajo en los dispositivos finales ni se corre riesgo de alteraciones por parte del cliente.

3.4 Agora

Agora [9] es la API que utilizaremos para añadir las videollamadas en tiempo real en la aplicación. Este servicio ofrece implementaciones para una gran cantidad de plataformas, pero lo más importante es que ofrece un paquete muy sencillo de utilizar para Flutter que permite implementar de forma eficiente estas llamadas en la aplicación. Para utilizar la API es necesario registrarse en su página oficial y conseguir un ID de aplicación que será necesario utilizar cada vez que iniciemos una llamada para que la API pueda identificar qué aplicación está utilizando el servicio.
La API es de pago, pero disponemos de 10000 minutos mensuales gratis, lo que permite tener un margen muy grande para realizar las pruebas de la aplicación sin necesidad de abonar nada. Una vez la aplicación entre en producción y haya suficientes reuniones como para superar los minutos gratuitos, el coste pasa a ser de 0.00399 $ por minuto de llamada, como muestra la figura 3.2.

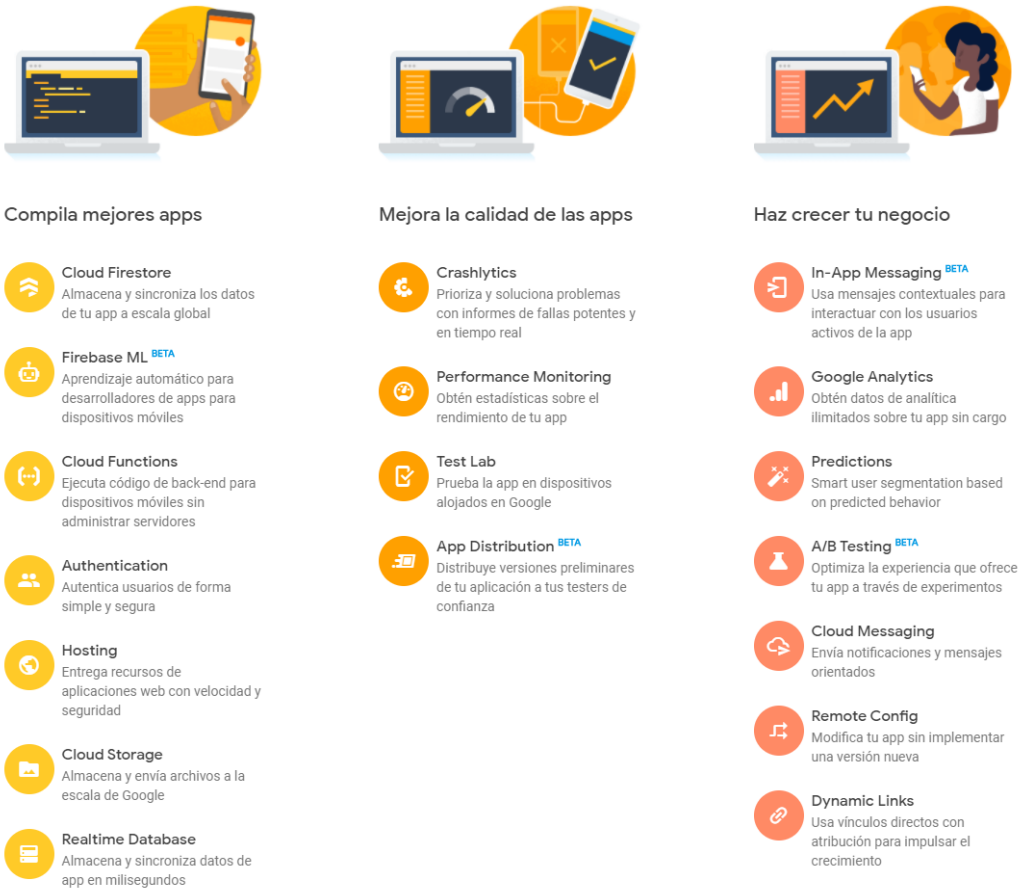

Figura 3.1: Funciones disponibles en Firebase.

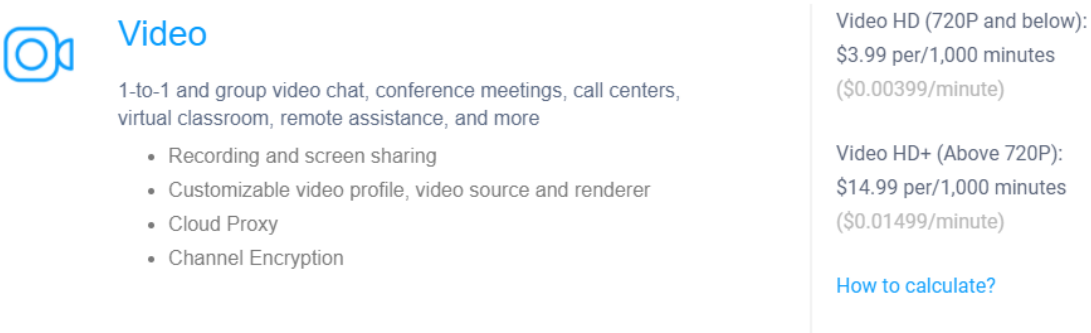

Figura 3.2: Precio en dólares de las videollamadas con Agora.

3.5 Aplicaciones para desarrollo y gestión

Estas aplicaciones y herramientas son las que se utilizarán para gestionar la linea temporal del proyecto, crear las tareas a realizar y anotar el diseño previo, diseñar su interfaz gráfica y escribir y compilar el código de la aplicación. También se incluyen las aplicaciones de control de versiones y Overleaf, utilizada para escribir esta memoria.

3.5.1 Sketch

Sketch [10] es un programa para edición de gráficos vectorial que se suele utilizar principalmente para implementar prototipos de interfaces de usuario, y que incluso se pueden ejecutar para simular un flujo de uso de lo que sería la aplicación final. Compite contra aplicaciones como Figma o AdobeXD. Sketch dispone de una amplia librería de *widgets* prediseñados que cualquier usuario puede descargar y utilizar en sus diseños, con la ventaja de que Google proporciona directamente un conjunto de elementos siguiendo la filosofía de Material Design, casi idénticos a los que incorpora Flutter para facilitar aún más su diseño y que este sea casi idéntico al de la aplicación final. Actualmente solo está disponible para su descarga en MacOs, y es necesario pagar una licencia de por vida de 99 dólares. Para este proyecto se utilizarán los 30 días de prueba gratuita que ofrece la herramienta.

3.5.2 Xcode

Xcode [11] es el entorno de desarrollo propietario de Apple y exclusivo para Mac destinado al desarrollo de aplicaciones para todos los dispositivos del ecosistema Apple. En nuestro caso, esta herramienta se utilizará para probar la aplicación en emuladores de iOS antes de probarla en dispositivos reales, pero no se editará código con ella.

3.5.3 Android Studio

Android Studio [12] es el entorno de desarrollo oficial para Android (reemplazando a Eclipse). Este entorno se utilizará para implementar toda la parte de Flutter, pues dispone de *plugins* para ello que permiten ejecutar la aplicación en emuladores o dispositivos reales con un solo clic, además de utilizar el Hot Reload que se explicó anteriormente. Android Studio también ayuda al desarrollador con sugerencias y predicciones de lo que este intenta escribir y te avisa de malas técnicas de programación o fallos y advertencias en el código.

3.5.4 Visual Studio Code

Visual Studio Code [13] es un editor de texto pero potenciado por la gran cantidad de *plugins* de los que dispone. Compite con otras alternativas como Sublime o Notepad++. Esta

herramienta se utilizará para implementar la parte de *backend* de Firebase, concretamente las funciones que se ejecutan en servidor, incluyendo el algoritmo de creación de grupos. Para ello, únicamente, será necesario el *plugin* de Dart. También se utilizará para editar algún archivo de configuración de la parte de iOS en caso de que sea necesario.

3.5.5 Taiga

Taiga [14] es una plataforma de gestión de proyectos con una interfaz simple y amigable. Contiene herramientas para organizar la línea temporal del proyecto, los casos de uso que se necesitan desarrollar, una Wiki con los datos que los participantes en el proyecto necesiten, y herramientas para desarrollos basados en Scrum que explicaremos con más detenimiento en el capítulo de metodología 4. Dependiendo del uso que se le de a la herramienta puede ser gratuita o tener un pequeño coste.

3.5.6 Miro

Miro [15] es un tablero colaborativo, que tiene una gran cantidad de usos posibles, ya que los participantes pueden crear esquemas, dibujos o anotar ideas en tiempo real. En este caso, se utilizará para el proceso de Design Sprint, que también se explicará más adelante. En caso de tener un máximo de tres tableros, el coste por equipo es gratuito.

3.5.7 Overleaf

Overleaf [16] es un editor de texto LaTeX colaborativo online, es una herramienta totalmente gratuita que permite compilar texto en formato LaTeX y mostrar el resultado de esta compilación al mismo tiempo que se edita el fichero. Se ha utilizado Overleaf para redactar esta memoria porque ofrece una gran facilidad para importar plantillas y compartir el proyecto con otros usuarios para que lo revisen. Además el documento está guardado en sus servidores y se puede editar desde cualquier navegador simplemente con iniciar sesión en tu cuenta.

3.5.8 GitLab

GitLab [17] es un servicio web de control de versiones basado en Git, se ha creado un repositorio para este proyecto pensando en subir compilaciones cada vez que se añade una funcionalidad nueva. Aunque en este caso al ser un proyecto individual no es tan necesario su uso porque el código no se comparte con otros desarrolladores, se puede utilizar para guardar versiones funcionales de la aplicación y tener la posibilidad de volver a ellas en caso de que una de las iteraciones cause algún error grave y sea necesario volver atrás en el tiempo. Para su uso no se ha utilizado la línea de comandos, si no que se han utilizado herramientas gráficas que facilitan un poco su uso, en este caso han sido Git Extensions para Windows y SourceTree

para MacOS, no se entrará en detalle pues hay una gran cantidad de herramientas de este tipo, su uso principal es realizar las mismas funcionalidades de la línea de comandos de Git pero con una interfaz más amigable.

Capítulo 4

Metodología

En este capítulo se explicarán los conceptos sobre el tipo de metodología escogida para realizar el proceso de desarrollo del software junto con las herramientas usadas para gestionar el ciclo de vida del desarrollo. La ejecución del proyecto se dividió en dos fases claramente diferenciadas: un *Design Sprint* y un desarrollo iterativo de tipo Scrum. El *Design Sprint* es de gran importancia ya que permite definir una gran cantidad de requisitos y casos de uso en un periodo de tiempo muy corto y, aunque su duración sea pequeña comparada con la duración del desarrollo Scrum, es un proceso de igual importancia. A continuación, se explicará en detalle cómo se ha llevado a cabo este *Design Sprint*.

4.1 Mínimo producto viable

El Minimal Viable Product (MVP) (del inglés *Minimal Viable Product*) o mínimo producto viable es un producto capaz de satisfacer las necesidades mínimas de un cliente. De esta forma, el cliente puede realizar pruebas con este producto, proporcionar cierto *feedback* para implementar cambios o mejoras en el producto si fuera necesario y verificar la viabilidad del mismo, como el propio nombre indica. El objetivo de este proyecto se centrará en obtener un MVP que cumpla con los objetivos/requisitos especificados, evitando de esa forma los costes que provocaría desarrollar un producto completo y permitiendo una mayor capacidad para realizar cambios si fuera necesario.

4.2 Metodologías ágiles

Para comenzar, es necesario conocer lo que son las metodologías ágiles para el desarrollo de software. Estas metodologías surgieron a partir de la necesidad de aumentar la capacidad de respuesta de los equipos de desarrollo ante los cambios y variantes que pueden surgir una vez ha comenzado el ciclo de vida de desarrollo del producto. Estas metodologías permiten

adaptar el ciclo de vida a las condiciones del proyecto de forma que, esta flexibilidad, ahorra muchos costes tanto de tiempo como monetarios y permite aumentar la productividad.

Las metodologías ágiles se guían por los siguientes doce principios que se recogen en el manifiesto Agile [18]:

1. La prioridad principal es la satisfacción del cliente a través de entregas tempranas de software con valor.
2. Se acepta que los requisitos cambien incluso en etapas tardías del desarrollo. Estos cambios proporcionan ventajas competitivas a los clientes.
3. Se entrega software funcional frecuentemente.
4. Los clientes y los desarrolladores trabajan juntos durante todo el proyecto.
5. Los productos se desarrollan en un entorno con trabajadores motivados, proporcionando las herramientas necesarias para ello.
6. Se recomienda tener reuniones periódicas presenciales, pues el método de comunicación más efectivo es el cara a cara.
7. El software funcionando es la principal medida de progreso.
8. Las metodologías ágiles promueven el desarrollo sostenible, es decir, que los participantes puedan mantener el ritmo de forma indefinida.
9. Atención continua a la excelencia técnica y al buen diseño para así mejorar la agilidad.
10. La simplicidad de cada tarea es esencial, siendo necesario dividir trabajos complejos en subtareas lo menos complejas posible.
11. Los equipos deben organizarse por sí mismos.
12. Se hará una revisión en intervalos regulares sobre como mejorar la forma en la que se trabaja para aplicarla a partir de ese momento.

4.3 Design Sprint

El *Design Sprint* es una metodología de desarrollo ágil, creada por Google, que permite prototipar y validar un proyecto junto a los clientes potenciales con el objetivo de definir cuáles serán las fases para su desarrollo. La idea inicial de *Design Sprint* contaba con cinco fases

de trabajo idealmente divididas en cinco días distintos y una fase anterior de preparación. En concreto, las fases que idealmente componen un *Design Sprint* son las siguientes:

- **Preparación:** consiste en reunir un equipo de personas, donde lo ideal sería contar con un número de entre cinco y siete personas, preferiblemente con ocupaciones distintas para que aporten puntos de vista distintos al proyecto. También es necesario pensar en una frase o el reto que intentará solucionar este *Design Sprint*.
- **Entendimiento:** esta es la fase que se realiza el primer día. La idea es que el equipo obtenga una misma base de conocimiento y que cada participante intervenga para exponer a los demás integrantes cuál es para él la mejor solución para abordar el problema.
- **Boceto:** posiblemente la fase más entretenida ya que, en este segundo día, los participantes tendrán que crear bocetos en papel que expresen gráficamente como verían ellos la solución que se va a implementar. Los bocetos pueden ser muy simples, pero lo importante es que estos aporten suficientes ideas.
- **Decisión:** el tercer día es en el que se toman las decisiones de como será el producto. La fase anterior devolvió como resultado una gran cantidad de ideas y el objetivo de este tercer día es reunir todas estas ideas, votar por cuáles son las mejores e intentar que se amolden entre sí.
- **Prototipo:** la fase cuatro consta de un único paso pero, posiblemente, el que requiere más conocimientos. Este paso consiste en crear un prototipo con alguna aplicación de creación de interfaces de usuario para que el posible cliente la valide. Es importante prototipar únicamente las funciones que, aparentemente, sea más importante validar y no centrarse en funciones básicas del producto. Una vez terminado este paso, también es interesante preparar las entrevistas que se deben realizar el último día.
- **Validación:** el último día de trabajo consiste en validar con el posible cliente el prototipo que resultó de la fase anterior. Es importante observar como interactúa el usuario con el prototipo sin aconsejarle para que este comportamiento sea lo más fiel posible al que tendrán otros clientes. Esta fase es importante para ver qué se puede cambiar y mejorar, y comenzar así el desarrollo del producto propiamente dicho.

Para realizar este proyecto se han adaptado las fases del *Design Sprint*, dado que el tiempo destinado para realizarlo eran tres días y no se disponía de clientes finales para validar el prototipo. Por ello, se realizaron tres sesiones de dos horas cada una, y el prototipo no será funcional ni tan detallado, únicamente mostrará de una forma más gráfica las funcionalidades

que se van a implementar. También hay que tener en cuenta que, debido a la situación actual, no se ha podido hacer de forma presencial. En la siguiente sección, se explica detalladamente como se ha adaptado esta metodología para este proyecto.

4.3.1 Fase de preparación

La fase de preparación se realizó antes de comenzar los tres días destinados al *Design Sprint*. Se decidió usar Miro como herramienta de apoyo ya que permite crear tableros colaborativos que todos los participantes puedan editar. De esta forma, la empresa que ya tenía experiencia en realizar este tipo de procedimientos creó una plantilla en Miro para ir rellenando durante el desarrollo del *Design Sprint*. Para participar en el proceso, a parte de mí (perfil de desarrollador), se unieron otras cinco personas pertenecientes a la empresa pero de distintos ámbitos, como pueden ser comerciales o de consultoría. Uno de ellos, la persona que tuvo la idea de comenzar con este proyecto, fue el que en las fases finales de votación tenía un voto con mayor peso. También se unió al proceso otra persona ajena a la empresa, también con perfil de desarrollador pero para otro tipo de tecnologías.

Lo siguiente fue definir el reto que se quería resolver. En este caso, ya se sabía de antemano que la solución sería una aplicación móvil aunque, normalmente, esta decisión se toma después de realizar todas las fases del *Design Sprint*. La frase elegida para representar el reto a resolver fue la siguiente: **Diseñar una solución que cubra el ciclo de vida del *networking* para maximizar las oportunidades de generar relaciones profesionales exitosas, reduciendo la barrera espacio-tiempo.**

Por último, la fase de preparación concluyó con la búsqueda de un par de personas a las que poder realizarles unas entrevistas cortas para pedir su opinión sobre el estado actual del ámbito en que se encuadra la aplicación.

4.3.2 Día 1: Entender

El día uno marca realmente el comienzo del *Design Sprint*. Todos los participantes se reunieron por videollamada y se conectaron al tablero virtual de Miro, que ya contenía la plantilla explicando los pasos a dar cada día y el tiempo destinado a cada uno.

El primer paso fue realizar las entrevistas a las personas con las que se contactó en la fase anterior. En este caso, las entrevistas se realizaron a dos personas y se marcó un tiempo de quince minutos para cada una de ellas. Es importante destacar que, en ningún caso, se les informó de que se estaba barajando la idea de implementar un producto. Las dos personas entrevistadas pertenecen a distintos ámbitos, una de ellas dirige proyectos en una importante

consultora informática y la otra es una persona muy activa que participa en eventos y sesiones de *networking* con mucha frecuencia. Esta diferenciación hizo que las entrevistas se enfocaran de forma distinta, y esto aportó una mayor cantidad de ideas al proyecto. En las entrevistas se preguntó a los voluntarios la forma en la que se relacionaban ellos y otras personas de su ámbito, mientras los participantes del *sprint* apuntábamos todas las frases que considerábamos relevantes de sus respuestas.

Una vez terminadas las entrevistas, cada participante recopiló en un mapa de empatía las frases que había apuntado, dividiéndolas en las siguientes zonas: **Alegrías, Frustraciones, ¿Qué escucha?, ¿Qué piensa y siente?, ¿Qué ve?, ¿Qué dice y hace?.**

No existe una definición exacta de qué frases hay que añadir a cada una de las zonas. Lo importante es que el mapa resultante ayude a organizar la gran cantidad de ideas que cada participante tiene apuntadas y, de esa forma, poder exponerlas a los demás participantes. Además, este tipo de mapas permiten ver que ideas que se han puesto en común y en las que no se está de acuerdo. En las figuras 4.1 y 4.2 se muestran las zonas de alegrías y frustraciones, respectivamente, del mapa de empatía relleno con los datos de la entrevista a la persona que asiste a eventos.

Figura 4.1: Mapa de empatía: alegrías.

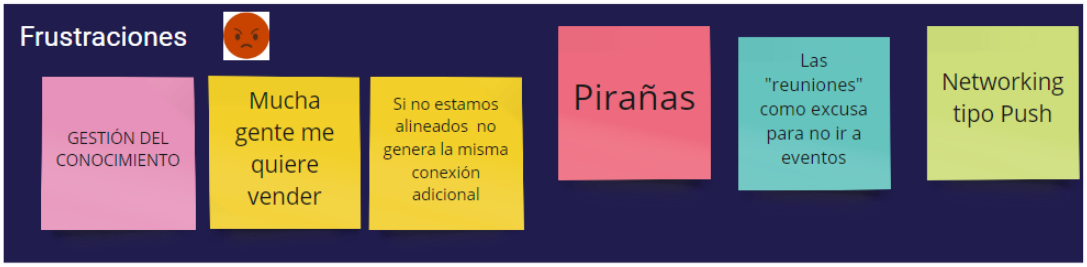

Figura 4.2: Mapa de empatía: frustraciones.

Para finalizar el primer día, con los participantes ya conscientes de los problemas a resolver y las opiniones de personas ajenas que no conocen la solución a implementar, llegó la hora de evaluar realmente los problemas a solucionar y ver cuál es el que puede marcar el éxito del producto. Los *How Might We (HMW)?/How might we?* o ¿Cómo podríamos...? son preguntas que intentan buscar una respuesta a los principales problemas que han surgido. En este caso, cada participante tuvo la libertad de escribir cuantos quisiera, pero dividiéndolos en diferentes secciones para clasificarlos mejor. En concreto, se tuvieron en cuenta las siguientes secciones: *Onboarding* o como introducir a los usuarios en la herramienta, *Engagement* o como potenciar su uso, *Match* o como conseguir generar relaciones y *Postmatch* o como aprovechar las relaciones generadas para que la aplicación conozca tus gustos.

Por último, se votaron los mejores *HMW?* para buscar cuál sería la principal idea a resolver y en la que se centraría el producto. En este caso lo más votado fue: **¿Cómo podríamos hacer un registro de usuario simple y no intrusivo, pero que aporte todos los datos necesarios del usuario?** En la imagen 4.3, se muestran los HMW? más votados.

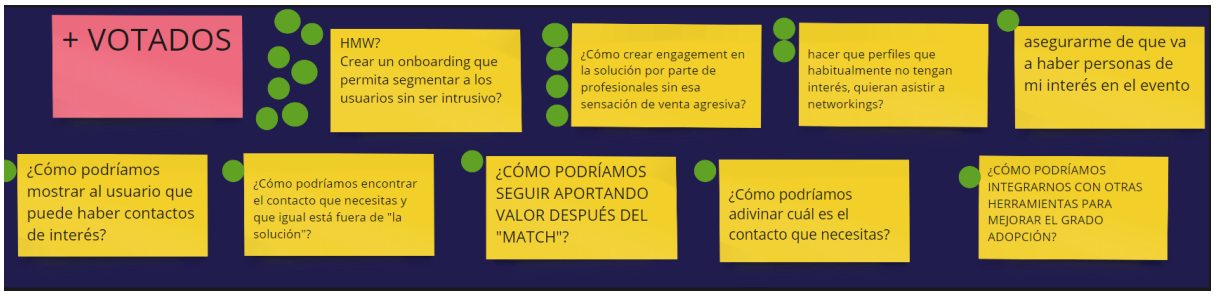

Figura 4.3: HMW? más votados.

4.3.3 Día 2: Idear

El segundo día se utilizó para que los participantes diéramos rienda suelta a nuestra parte creativa. Para comenzar, todos los participantes tenían que preparar tres aplicaciones de la competencia que resolvieran estos problemas o que aportaran funcionalidades que fueran de utilidad. Para ello, era necesario añadir en un espacio dedicado del tablero unas capturas de pantalla de estas herramientas acompañadas de pequeñas explicaciones anotadas. Una vez hecha esta investigación, cada participante tuvo unos minutos para explicar a los demás porqué eligió esas funcionalidades y lo que aportarían a la nueva aplicación.

A continuación, una vez analizadas todas las funcionalidades insertadas en el tablero, hubo un periodo corto de cinco minutos para escribir que funcionalidades creíamos que la aplica-

ción debería tener obligatoriamente, sin importar si estas eran viables o no. Es importante destacar que, hasta este momento, no se comenzaron a desarrollar nuevas ideas en la aplicación ya que toda la parte anterior del *Design Sprint* se centró más en investigación y búsqueda de posibles soluciones.

Para finalizar con el segundo día, lo siguiente fue un divertido juego:"Desvarío en 8". Simplemente consiste en que cada participante tiene que coger una de las ideas de funcionalidad que se escribieron en el paso anterior y dibujar ocho diseños distintos para esa funcionalidad dentro de la aplicación, pero con la complicación de que sólo se dispone de un minuto para diseñar cada boceto. Estos bocetos son privados y no se comparten con los demás compañeros, por lo que la calidad del dibujo no importa, lo importante es, como dice el nombre del juego, desvariar y encontrar interfaces innovadoras para cubrir la funcionalidad.

4.3.4 Día 3: Decidir

El último día fue quizás el más importante de todos ya que se votaron las decisiones más importantes en cuanto a la forma en la que el usuario interactuaría con la aplicación y la estética que esta tendría.

Para comenzar, cada participante debía de tener preparados en el tablero unos diseños de cómo se verían algunas de las funcionalidades de la aplicación. Posiblemente, estas pantallas deriven del juego de "desvarío" del día anterior. No había unas reglas explícitas de como realizar estos diseños, por lo que algunos decidieron realizarlos en papel, mientras que otros se decantaron por hacerlos con alguna herramienta de creación de *mockups*.

Una vez preparados los diseños, cada participante hizo una exposición detallada de las funcionalidades que había diseñado y el porqué de estas. En este momento es donde, verdaderamente, fue útil la diversidad de los participantes ya que, tener personas de distintos ámbitos profesionales y diversas edades, hizo que cada persona se centrara en cosas totalmente distintas. Algunos priorizaron un diseño limpio y diseñaron las pantallas de inicio y de reuniones. Otros participantes, con un diseño menos cuidado, decidieron centrarse en el registro de los usuarios y dar una buena explicación en palabras de como debería de funcionar. En resumen, cada uno aportó ideas buenas y distintas.

Por último, fue necesario realizar dos tipos de votaciones. En la primera, los participantes debían decidir cuales de todas las funcionalidades presentadas les parecían de mayor importancia. La segunda votación permitió decidir cuál de los diseños sería el que se tomaría como

referencia para realizar el prototipo, centrándose en la parte meramente estética y de facilidad de uso. En la figura 4.4 se muestran tres de los *mockups* más votados.

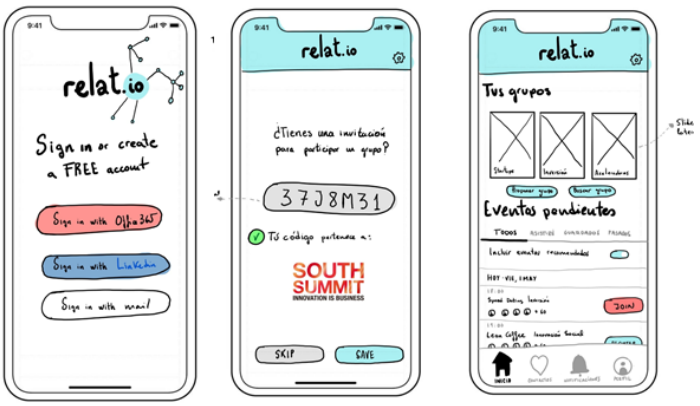

Figura 4.4: Ejemplo de mockups más votados.

El siguiente paso de este día fue que cada participante construyera un flujo de prueba del usuario, es decir, los pasos por los que tendría que pasar idealmente el usuario para alcanzar el objetivo principal de la aplicación. En este caso, el objetivo principal es hacer un contacto. Este flujo se dividió en seis pasos y, una vez finalizado, los participantes votaron cuál de todos estos flujos era el mejor.

El último paso de todo el *Design Sprint* fue sencillo ya que, simplemente, se deben asociar los diseños presentados en el primer paso de este día con el flujo de usuario diseñado en el paso anterior. Esta asociación nos permite tener una idea más clara de como sería visualmente cada uno de los pasos definidos en el diagrama de flujo.

4.3.5 Conclusiones

El *Design Sprint* fue diseñado originalmente para demostrar la viabilidad de un producto antes de empezar a desarrollarlo. En el caso particular de esta aplicación, no se disponía del tiempo para realizar todos los pasos ni de un cliente final para validar un prototipo funcional, por lo que no fue un proceso ideal para demostrar si la aplicación se podría vender en un futuro. Sin embargo, el objetivo en este caso no era ese, sino que se buscaba aclarar las ideas de como tendría que ser la aplicación antes de comenzar con un proceso de maquetado realista, cosa que se consiguió sin ninguna duda. Además, las personas entrevistadas el primer día sí que mostraron una necesidad o interés por una herramienta de este tipo, teniendo en cuenta que en ningún momento se les mencionó que se estaba pensando en desarrollar una.

Como aspectos a mejorar, sería interesante realizar este proceso de la forma en la que se pensó originalmente, con cinco sesiones de trabajo e intentando diversificar aún más a los participantes con el objetivo de comprobar hasta que punto se puede mejorar la idea a desarrollar en un periodo de tiempo tan corto.

4.4 Scrum

Scrum en un marco de trabajo que ofrece una serie de reglas y herramientas para facilitar el proceso de desarrollo software. De hecho, es uno de los más utilizados en los últimos años. Aunque se suele utilizar para equipos de desarrollo, su uso se está extendiendo porque también se puede aplicar a todo tipo de trabajos en equipo. Para comprender totalmente Scrum, primero es necesario conocer los conceptos relacionados con las personas, elementos y acciones que intervienen en este proceso. Se definen a continuación.

4.4.1 Roles

Definen el rol y el tipo de trabajo que tendrá que realizar cada participante del equipo. Los principales roles son:

- **Product Owner:** Es el portavoz del cliente, encargado de gestionar el *product backlog*. Trabaja codo con codo con el cliente y el equipo Scrum para transmitir sus ideas, y de el depende en gran parte que el proyecto se desarrolle con éxito.
- **Scrum Master:** Aunque el nombre parezca que impone un rol de mandato, el Scrum Master es un líder al servicio del equipo Scrum, pero no tiene autoridad jerárquica sobre este. Su rol se basa en ayudar al equipo eliminando impedimentos y tareas. Además, debe participar en las reuniones y asegurarse de que se cumplen los objetivos temporales.
- **Equipo Scrum:** Normalmente compuesto por tres a nueve miembros, es un equipo multidisciplinar en el cual debe haber una confianza total entre ellos y un reparto constante del trabajo y los datos de este. Es el encargado de desarrollar el producto en sí.

4.4.2 Eventos Scrum

Son eventos predefinidos y organizados en el tiempo de antemano. Se utilizan para aumentar la productividad y reducir la longitud de las reuniones necesarias. Estos eventos son:

- **Sprint:** Es el nombre que recibe una iteración en el marco de desarrollo Scrum. Su duración suele ser de entre una semana y dos meses, y permite tener un ritmo de trabajo

prefijado. Lo ideal es que cada *Sprint* tenga una complejidad similar y que se pueda mantener el ritmo durante un tiempo indefinido. Una vez terminado cada *Sprint*, el producto que se entrega debe ser totalmente funcional y testeable.

- **Sprint planning:** Básicamente consiste en una planificación de cuál va a ser el siguiente *Sprint* a realizar, analizando el resultado del anterior y organizando temporalmente el siguiente. La duración de estos eventos suele estar prefijada dependiendo de la duración del *Sprint*.

- **Daily:** Como su nombre indica, es una reunión diaria en la que participa el equipo Scrum. Su objetivo es poner al equipo en contexto y aumentar el compañerismo averiguando cuáles son los siguientes pasos, los problemas que se han tenido y cómo se pueden resolver.

- **Revisión de Sprint:** Puede integrarse dentro del *Sprint Planning*, y consiste en una reunión informal para analizar los casos de uso que se han implementado y hacer cambios en estos o en los siguientes en caso de ser necesario. Es útil para ir revisando el estado del producto por parte del *Product Owner*.

4.4.3 Mecanismos de transparencia de Scrum

Los mecanismos de transparencia, como su propio nombre indica, están diseñados para aumentar la transparencia y permitir a los trabajadores comprender mejor lo que pasa en el proyecto. Entre ellos, destacan:

- **Product backlog:** Es el resultado del trabajo del *Product Owner* con el cliente, y contiene un listado de todos los casos de uso a implementar para llegar al producto final. Su tamaño disminuye normalmente a medida que avanza el proyecto, pero se pueden añadir elemetos en el caso de que surja un nuevo requisito o un *bug*.

- **Sprint backlog:** Contiene todo el trabajo que se va a realizar en un *Sprint*, y su contenido deriva del contenido del *Product Backlog*. La mejor forma de representarlo es con un Kanban, que se explica posteriormente.

- **Épicas:** Son grandes cantidades de trabajo que se pueden desglosar en subtareas donde, normalmente, la tarea mínima es lo que se conoce como historia de usuario. Las épicas pueden ser útiles para mostrar cómo está siendo el avance en cada una de las grandes funcionalidades de la aplicación.

- **Incremento:** El incremento es la parte que se entrega al finalizar un *Sprint*, debe de estar terminado y ser funcional.

- **Kanban:** También se considera una metodología de desarrollo ágil, pero en Scrum se puede adaptar una de las características de Kanban para mejorar el método. El Kanban es un tablero que contiene las tareas a realizar en un *Sprint*, pero ordenadas por columnas de forma que cada columna representa un estado de estas tareas. Un ejemplo de estados sería el siguiente: no listo, listo, en progreso, listo para test, finalizado.

- **Historias de usuario:** son descripciones de los casos de uso o requerimientos del cliente que ayudan al equipo de desarrollo a comprender mejor como tiene que ser el resultado de una tarea. El esquema más utilizado es el siguiente: **Como <quién> Quiero <qué> Para <objetivo>**, siendo "quién" el actor, "qué" el caso de uso que se quiere realizar, y "objetivo" la forma en la que se quiere realizar. Se mostrará algún ejemplo con más detalle en la siguiente sección.

- **Spikes y errores:** son dos tipos de historias de usuario especiales que se añaden al *Product Backlog*, normalmente durante el desarrollo o test de una funcionalidad. Un *Spike* es una "pausa" y se utiliza cuando hay algún problema para desarrollar y se necesita un tiempo para su investigación. Un error, como su nombre indica, se introduce cuando es necesario solucionar algún fallo detectado de un *Sprint* anterior.

4.5 Scrum dentro de este proyecto

Para desarrollar este proyecto, se ha usado el marco de desarrollo Scrum explicado en la sección anterior, pero modificándolo pues no hay un cliente en sí y el equipo de desarrollo está formado por una sola persona. De todas formas, se ha intentado seguir de la forma más fiel posible esta metodología.

Con ayuda de la persona que tuvo la idea de desarrollar esta aplicación, se rellenó el *Product Backlog* con las historias de usuario necesarias para el MVP y algunas más que, en principio, no sería necesario implementar pero que se consideran para su desarrollo posterior. Esta persona también ha sido la encargada de valorar los resultados de cada *Sprint* participando en las revisiones al final de cada uno de ellos y en revisiones en medio de estos para mostrar los avances. Los *Sprints* finalmente han sido de una semana de duración, reservando una parte de las mañanas de los viernes para realizar la revisión final y el *Sprint Planning* para la semana siguiente.

En la planificación inicial, se establecieron un total de once *Sprints* para desarrollar todas las historias de usuario para el MVP o, lo que es lo mismo, se planificó una duración total del proyecto de once semanas. En el capítulo de planificación 6 se entrará más en detalle.

4.5.1 Implementación de Scrum en la práctica usando Taiga

Taiga ha sido la herramienta utilizada para organizar el desarrollo del proyecto. En esta sección, se muestran ejemplos de cómo ha sido utilizada y las opciones que tiene para ayudar al equipo de desarrollo.

Epicas

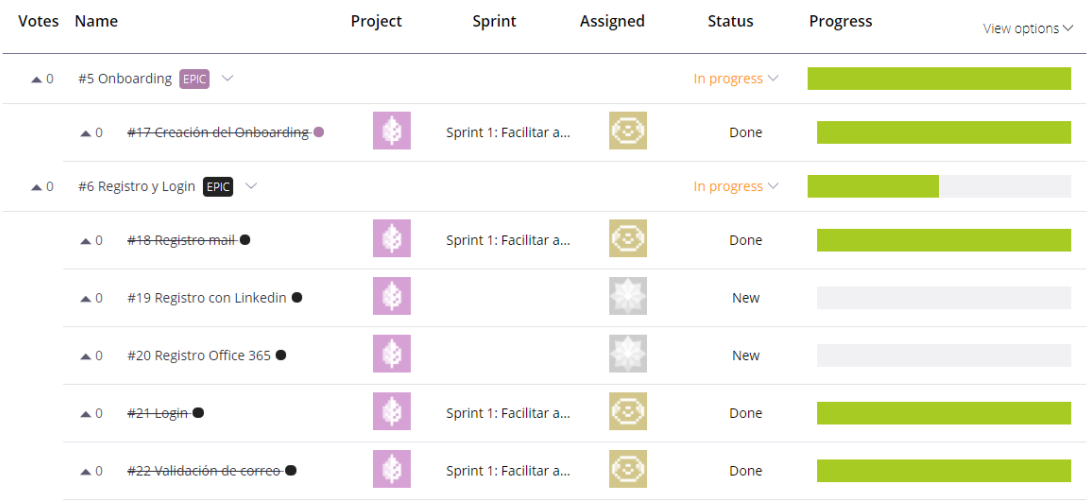

Figura 4.5: Ejemplo de las épicas de *onboarding*, registro y login.

En Taiga podemos definir una serie de épicas que agrupan las historias de usuario en categorías más grandes. Esto es útil pues la herramienta muestra datos sobre el porcentaje de avance de una épica, el estado de esta (nueva, en progreso o finalizada), cada una las historias de usuario que la forman y el estado de cada una de ellas. En el momento en el que todas las épicas tengan un porcentaje de realización del 100%, significará que todas las historias de usuario han sido implementadas y, por lo tanto, la aplicación está terminada. En la figura 4.5 se muestra el ejemplo de dos épicas definidas en Taiga, con el porcentaje restante para completarlas.

Backlog

En el *backlog* de Taiga tenemos, en forma de tabla, todas las historias de usuario que faltan por implementar. Además, con esta utilidad podemos asignarle a cada una de ellas el *Sprint* en el que queremos que se implementen. Otra característica interesante es que tiene la opción de que el equipo Scrum vote cuáles deberían de ser las siguientes historias de usuario a implementar, para tenerlo en cuenta en el siguiente *Sprint Planning*. En la imagen 4.6, tenemos un ejemplo del *backlog* con las historias que han quedado sin implementarse una vez terminado el MVP.

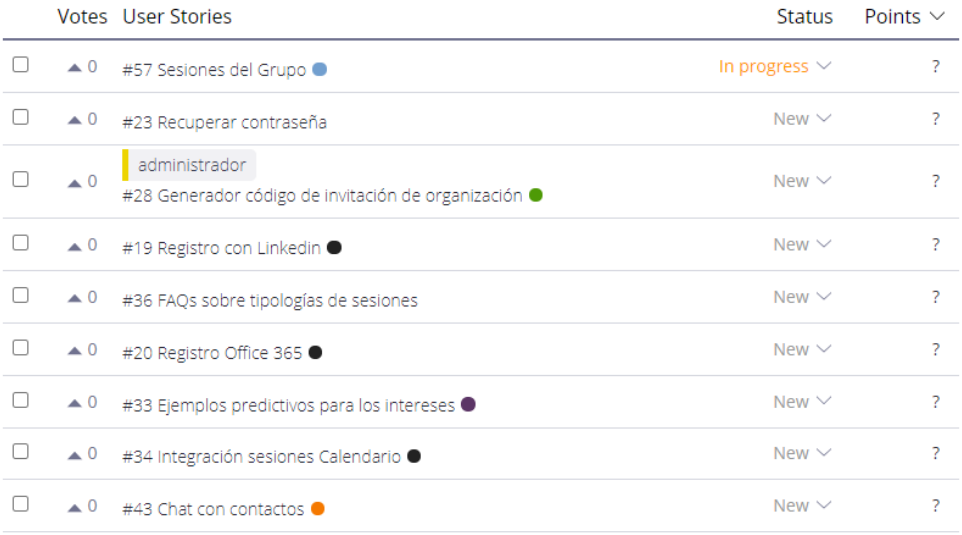

Figura 4.6: Ejemplo del *backlog* una vez finalizado el MVP.

Kanban

El Kanban es un tablero que se rellena con todas las historias de usuario del proyecto. La idea es mover cada historia de usuario a la columna que indica el estado en el que se encuentra. Por ejemplo, en el caso de Taiga, los estados disponibles son: nuevo, listo, en progreso, listo para probar, finalizado y archivado. De esta forma, el Kanban es un mecanismo de transparencia muy bueno para que cualquier persona pueda seguir de forma rápida y sencilla el progreso de cada *Sprint*.

En este proyecto, se ha utilizado este tablero para colocar en estado "listo" las historias de usuario que se realizarán en cada *Sprint* y que ya se han explicado en la reunión de planificación. Una vez se comienza con la implementación de una historia, esta pasa a estado en progreso y, una vez terminada, se pone en estado listo para prueba. En el momento en que, en la reunión de revisión del *Sprint*, se confirme que el desarrollo es correcto, esta pasa a finalizada. Taiga finaliza automáticamente un *Sprint* cuando todas sus historias de usuario están finalizadas, por lo que es no es necesario modificar explícitamente la información del *Sprint* correspondiente. En la figura 4.7, se muestra un ejemplo de un Kanban con cinco estados.

Historias de usuario

Es la unidad mínima en Taiga, y un conjunto de ellas forman una épica. A cada historia de usuario se le asigna el esquema de **Como <quién> Quiero <qué> Para <objetivo>**. Además, tenemos la opción de añadir criterios de aceptación para saber si se cumplen todas las condiciones, insertando en ellas instrucciones o imágenes. Otra funcionalidad interesante que

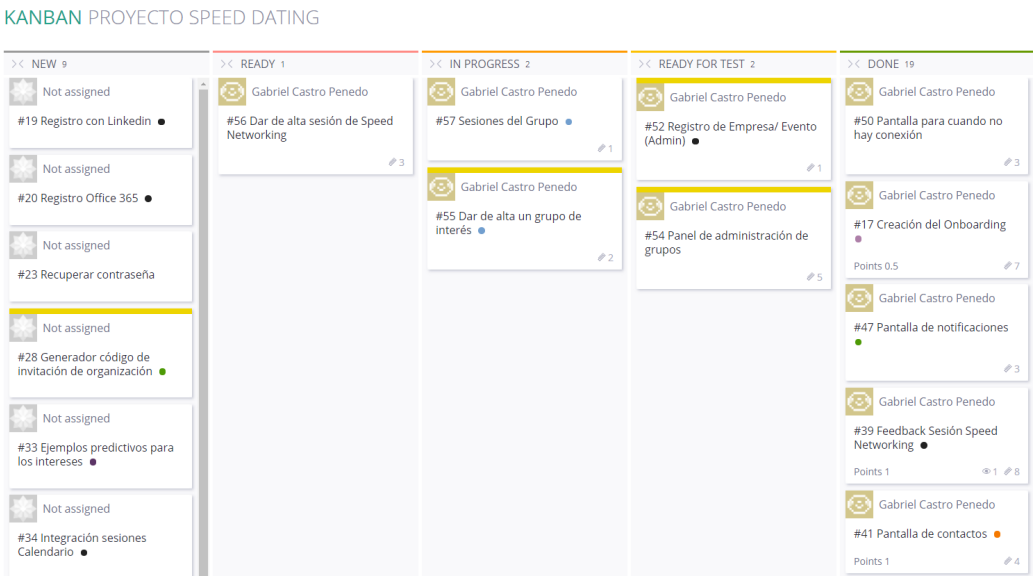

Figura 4.7: Ejemplo de un Kanban con historias de usuario en todos los estados.

ofrece Taiga es la posibilidad de asignar puntos de dificultad a estas, lo que permite que a cada *Sprint* se le asignen historias de usuario que sumen la misma puntuación aproximada y, así, nos aseguramos de que el nivel de dificultad se mantenga en cada *Sprint*. En la figura 4.8, se muestra la historia de usuario de login.

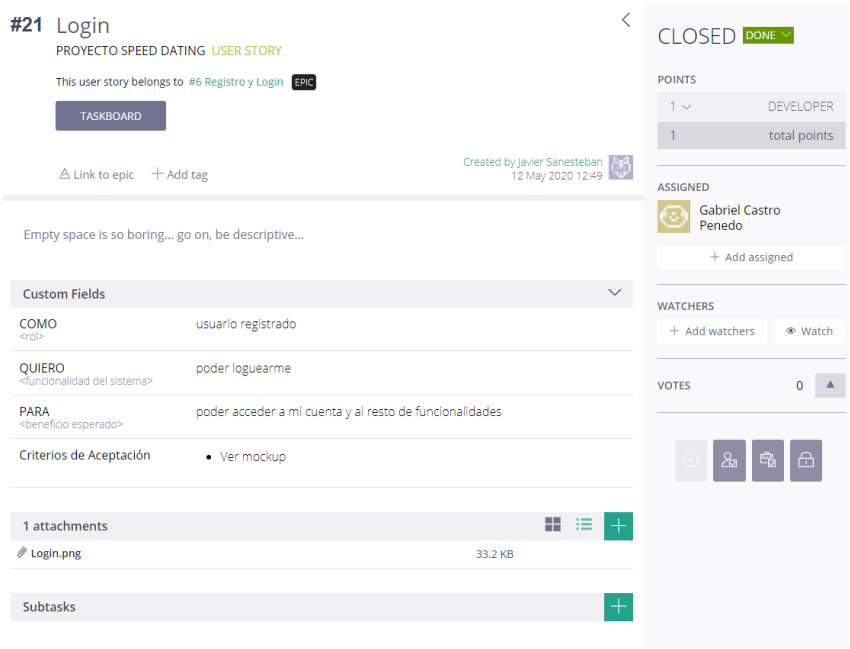

Figura 4.8: Ejemplo de la historia de usuario de login.

Capítulo 5

Análisis de requisitos

En este capítulo se explican los requisitos de la aplicación, los actores que interactúan con en ella y los casos de uso que se implementarán para realizar el MVP.

5.1 Actores

- **Administrador:** es la persona encargada de administrar los grupos de interés y sesiones que se organizan dentro de un evento/empresa. Para ello, tendrá que tener permisos para identificarse en la aplicación de administración. Para el MVP, simplemente, podrá crear grupos y crear sesiones dentro de ellos. Cada administrador, por el momento, tendrá una cuenta distinta para cada evento.

- **Usuario no registrado:** un usuario sin registrar no tendrá acceso a ninguna de las funcionalidades principales, únicamente podrá crear su cuenta y ver una pantalla de presentación sobre las funcionalidades de la aplicación.

- **Usuario registrado:** Una vez un usuario se ha registrado, tendrá acceso a los eventos a los que esté apuntado. Este actor podrá suscribirse a grupos de interés, inscribirse en las sesiones de estos, participar en videollamadas con personas con intereses similares a él, tener una agenda de contactos, notificaciones sobre sus nuevos contactos y nuevos grupos de interés y editar su perfil con su experiencia, estudios, descripción e intereses.

5.2 Requisitos funcionales

Los requisitos funcionales definen comportamientos o funciones que debe implementar el sistema. Además, pueden declarar explícitamente lo que el sistema no debe hacer. Los requisitos funcionales definidos para esta aplicación se resumen en los siguientes puntos:

- **Login y registro de usuario:** Permite que cualquier usuario se registre y complete su perfil aún sin pertenecer a ningún evento o empresa.
- **Editar perfil:** cualquier usuario puede editar datos de su perfil, como foto, intereses, experiencia laboral, etc.
- **Ver grupos de interés:** los usuarios que hayan sido invitados a algún evento pueden ver información y apuntarse a cualquier grupo de interés.
- **Ver sesiones:** los usuarios pueden ver un calendario con las sesiones creadas para sus grupos de interés y los detalles de estas sesiones para poder inscribirse o descartarlas.
- **Ver contactos:** los usuarios pueden ver una agenda con los contactos realizados.
- **Ver notificaciones:** los usuarios recibirán notificaciones sobre eventos y contactos.
- **Participar en sesiones:** los usuarios podrán participar en sesiones con videollamadas agrupados por intereses. Cuando un usuario se inscribe en una sesión dentro de sus grupos de interés, la aplicación se encarga de organizar rondas rápidas de videollamadas en las que el usuario participa con distintos grupos de personas con los que comporte ciertos intereses. Al final de esta ronda de reuniones, puede valorar a los demás participantes y guardar los contactos de forma automática en caso de que valoración positiva en ambos sentidos.

5.3 Requisitos no funcionales

Los requisitos no funcionales describen atributos que no afectan al comportamiento del sistema, sino a su calidad de funcionamiento. Entre ellos, se pueden destacar los siguientes:

- La aplicación debe de tener un estilo profesional y diferencial, no debe parecerse a aplicaciones básicas para que la sensación a primera vista sea de una aplicación seria y orientada a un entorno profesional.
- La calidad de las llamadas debe ser estable para que el usuario no dude de su funcionamiento.
- La resolución de vídeo de las llamadas debe ajustarse para buscar una buena relación entre la calidad visual del vídeo y el consumo de datos móviles de este.
- Se deben precargar datos para que las transiciones entre pantallas sean fluidas y sin tiempos de carga.

- El tiempo de ejecución del algoritmo de agrupamiento debe ser menor a veinte segundos para que el usuario no tenga la sensación de esperas largas o de que la aplicación se ha caído.

5.4 Casos de uso

En esta sección, se enumeran los casos de uso que se implementarán para el MVP. Estos casos de uso derivan directamente de las funcionalidades que se decidieron implementar tras la realización del *Design Sprint*. Existen otros casos de uso que, aunque también es posible que se implementen más tarde, no son necesarios para probar la viabilidad de la aplicación por lo que quedan en segundo plano y no se explicarán en este capítulo.

5.4.1 Administrador

Estos son los casos de uso que se implementarán en un principio para la aplicación web de administración:

- **Pedir permisos para ser administrador:** en un principio, habrá que enviar un correo para que a un usuario se le concedan permisos de acceso de administración a la aplicación.
- **Iniciar sesión como administrador.**
- **Ver una lista de los grupos de interés:** el administrador podrá ver un listado de los grupos de interés de la empresa/evento que gestiona.
- **Añadir un nuevo grupo de interés:** se podrá crear un nuevo grupo de interés en el evento.
- **Guardar un nuevo grupo de interés como borrador:** se podrá guardar un grupo de interés pero sin publicarlo a los usuarios.
- **Ver las sesiones de un grupo de interés:** dentro de los detalles de un grupo de interés se podrán ver las sesiones creadas.
- **Añadir una sesión dentro de un grupo de interés:** se podrán crear nuevas sesiones dentro de un grupo de interés.
- **Cerrar sesión.**

5.4.2 Usuario

Estos casos de uso corresponden a la aplicación móvil que utilizaran los usuarios finales:

- **Registrarse.**
- **Iniciar sesión.**
- **Confirmar correo electrónico:** una vez se ha registrado un usuario, le llegará un correo para confirmar el registro.
- **Añadir o editar datos de perfil:** durante el primer registro o en la página de perfil se podrán editar los datos sobre el perfil de un usuario.
- **Añadir, editar o eliminar foto de perfil:** durante el primer registro o en la página de perfil se podrá editar la foto de perfil.
- **Añadir, editar o eliminar experiencia laboral:** durante el primer registro o en la página de perfil se podrá editar la experiencia laboral.
- **Añadir, editar o eliminar estudios:** durante el primer registro o en la página de perfil se podrán editar los estudios.
- **Configurar gustos:** durante el primer registro o en la página de perfil se podrán editar los intereses que, más tarde, serán utilizados por la aplicación para emparejar al usuario en las reuniones.
- **Ver listado de grupos de interés:** en la pantalla principal se podrá ver un listado de los grupos de interés de los eventos para el usuario.
- **Ver detalles de un grupo de interés.**
- **Suscribirse o cancelar suscripción de un grupo de interés.**
- **Ver próximas sesiones:** se podrá acceder a un calendario con las sesiones de los grupos de interés a los que se está suscrito.
- **Ver detalles de una sesión.**
- **Inscribirse o cancelar inscripción a una sesión:** en la pantalla de detalles de sesión se podrá confirmar la asistencia a esta o cancelarla en caso de ya estar inscrito.
- **Ordenar sesiones por fecha o inscripción:** en el calendario de sesiones, se podrán ordenar estas de dos formas, por fecha de inicio (de más cercana a más lejana), o por inscripción (primero las sesiones en las que el usuario ya está inscrito).

- **Ver listado de contactos realizados.**
- **Ver listado de notificaciones:** ver un listado de las últimas notificaciones ordenadas por fecha.
- **Eliminar notificaciones del listado.**
- **Cerrar sesión:** El usuario podrá cerrar sesión para salir de su cuenta y volver a la pantalla de *login*.
- **Entrar en sala de espera de sesión:** diez minutos antes de que empiece una sesión a la que se está suscrito, se podrá entrar en una sala de espera.
- **Salir de una sesión:** en cualquier momento se podrá abandonar una sesión.
- **Agrupar por intereses:** el usuario que participa en una reunión será agrupado por intereses con el resto de participantes.
- **Participar en videollamada:** una vez empiece una sesión, se participará en videollamadas con otros participantes agrupados por afinidad.
- **Silenciar micrófono en videollamada:** dentro de una llamada, se podrá silenciar el micrófono y los demás participantes sabrán si lo has silenciado.
- **Colgar videollamada:** dentro de una llamada, se podrá colgar antes de que termine el tiempo, el usuario pasará directamente a la pantalla de votaciones y esperará a que terminen el resto de salas.
- **Cambiar cámara de videollamada:** dentro de una llamada, se podrá cambiar entre la cámara principal o delantera del dispositivo.
- **Puntuar usuarios de una reunión:** una vez finalizada una llamada dentro de una sesión se podrá puntuar a los otros participantes.
- **Ver contactos hechos en una reunión:** una vez terminada la sesión se mostrarán los nuevos contactos en caso de que se hayan realizado.

Capítulo 6

Análisis y planificación económica

En este capítulo se analizarán los requisitos y la planificación previa al comienzo del desarrollo, el coste estimado de los recursos utilizados tanto a nivel humano como de material, y se explicará la organización inicial en *Sprints* que se ha planeado. Por último, se comparará la planificación inicial con el desarrollo real del proyecto.

6.1 Recursos necesarios

En esta sección se exponen todos los recursos que fueron necesarios para la realización del proyecto, tanto humanos como materiales.

6.1.1 Humanos

Aunque el desarrollo se basó en un marco de tipo Scrum, en el proyecto únicamente participaron dos personas:

- **Product Owner:** en este caso se trata de la persona que tuvo la idea de la aplicación. Sus competencias son ayudar a rellenar el *Product Backlog*, participar en las revisiones y planificaciones de *Sprints*, y ayudar a solucionar problemas o dudas del desarrollador, por lo que se puede decir que su rol es realmente el de jefe de proyecto, *Product Owner* y cliente al mismo tiempo.

- **Equipo de desarrollo:** en este caso el equipo Scrum está formado únicamente por el propio alumno. Sus competencias son ayudar a rellenar el *Product Backlog*, participar en las revisiones y planificaciones de los *Sprint*, diseñar la interfaz de la aplicación, desarrollar el código de la aplicación, probarlo y documentar, por lo que su rol puede ser el de *Scrum Master*, analista, diseñador y desarrollador al mismo tiempo.

6.1.2 Hardware

El hardware o recursos materiales utilizados durante el ciclo de vida del proyecto son los siguientes:

- **Ordenador MacOS:** se ha elegido desarrollar la aplicación en un ordenador con sistema operativo MacOS pues es necesario para realizar las compilaciones para dispositivos de Apple (se ofrece una alternativa en el anexo B) y no ofrece impedimentos para compilar para Android o Web. Además, este sistema operativo ofrece la capacidad de probar la aplicación en emuladores de Android o iOS sin ser necesario instalarla en un terminal real. En caso de haber utilizado un dispositivo con Windows o Linux, no se habría podido desarrollar la aplicación para teléfonos Apple.
- **Teléfono móvil con Android:** será el dispositivo que se use con más frecuencia para probar la aplicación y asegurarse de que la interfaz es usable en un dispositivo real, pues con los emuladores se puede probar el funcionamiento de la aplicación pero la experiencia de usuario no se aprecia de la misma forma. El dispositivo corría la versión beta de Android 11.
- **Teléfono móvil con iOS:** dado que la aplicación está destinada a su uso en los dos sistemas operativos móviles más utilizados, y que el comportamiento de iOS no es el mismo que el de Android en términos de navegación (por ejemplo, iOS no dispone de un botón atrás, por lo que hay que insertarlo en pantalla), se ha utilizado un dispositivo iOS para realizar algunas pruebas periódicas y comprobar que todo funciona bien en ambas plataformas. El dispositivo corría iOS 13.

6.1.3 Software y servicios

El software o servicios que se han utilizado durante el desarrollo de la aplicación son los siguientes:

- **Aplicaciones de diseño:** para hacer los diseños de la aplicación se ha usado la versión de prueba de Sketch, que ofrece treinta días gratuitos. En caso de que se use un mayor periodo de tiempo, la licencia tiene un cierto coste por lo que se va a tener en cuenta en la planificación económica.
- **IDEs:** Las aplicaciones con las que se ha programado el código de la aplicación como Android Studio, VSCode o Xcode son totalmente gratuitas, aunque esta última requiere obligatoriamente un dispositivo con MacOS.
- **Servicios y API:** Durante el desarrollo se han aprovechado los planes gratuitos de Firebase y los minutos gratuitos que ofrece Agora para las videollamadas. En el momento

en el que la aplicación salga al mercado, en caso de tener éxito, es posible que se superen los límites de la suscripción gratuita. Sin embargo, esto no se tendrá en cuenta pues es un coste derivado del mantenimiento/comercialización posterior a la entrega del producto.

- **Licencia para tiendas de aplicaciones:** para subir la aplicación a las tiendas de Android e iOS será necesario registrarse como desarrollador en ambas y esto también implica un coste adicional.
- **Aplicaciones de gestión:** tango Taiga como Miro tienen planes gratuitos y, por tanto, no será necesario añadir ningún coste adicional.

6.2 Planificación temporal inicial

Para explicar la planificación temporal es necesario aclarar que el equipo de desarrollo, en este caso el alumno, trabaja a media jornada de lunes a viernes, es decir, que una duración de una semana se traduce aproximadamente en veinte horas de trabajo real. Además, hay que tener en cuenta que estas horas se refieren estrictamente a tiempo de "desarrollo", es decir, no incluyen el trabajo de aprendizaje de las tecnologías ni el tiempo de documentación.

En esta planificación inicial, se detallan las historias de usuario que se implementarán en cada *Sprint* que, como ya se ha comentado, tendrán una duración fija de una semana. Estas historias de usuario vienen determinadas por el diseño realizado en el *Sprint* "número cero" que, además, es el único con dos semanas de duración. De esta forma, el desarrollo del proyecto se ha organizado en un *Design Sprint*, un *Sprint* "número cero" centrado en el diseño visual de la aplicación y un total de 11 *Sprints* para la implementación.

6.2.1 Design Sprint

Los *Design Sprint* tienen siempre un tiempo predefinido, pues uno de los requisitos para realizar un buen *Design Sprint* es el de controlar los tiempos de una forma lo más ajustada posible. Para este proyecto, la adaptación realizada para el *Design Sprint* duró tres días, con dos horas de trabajo por día, a lo que también hay que sumar una reunión final en la que se establecieron los objetivos a corto plazo, y se votó por el nombre de la aplicación y el esquema de colores que se utilizaría para el diseño de la interfaz.

6.2.2 Diseño de las interfaces

Se puede considerar como el *Sprint* 0, previo al comienzo de la fase puramente de desarrollo del proyecto. Para este *Sprint*, se reservan dos semanas con el objetivo de realizar un

diseño realista de cómo tendría que verse la interfaz de la aplicación y revisarlo con el *Product Owner*. Se diseñó tanto la aplicación móvil como la aplicación web de administración, con el objetivo de ser realista e incluir todas las funciones de la aplicación final en el diseño aunque estas no se fueran a implementar para el MVP. De esta forma, si en algún momento antes de que se termine el desarrollo surge algún cliente potencial, se le podrá enseñar de forma aproximada como será la aplicación y qué puede esperar de ella.

6.2.3 Sprint 1

En este *Sprint* se abordan las historias de usuario correspondientes al *onboarding*, registro y login. En concreto, se implementarán las siguientes historias de usuario:

Creación del onboarding

COMO <usuario no registrado> QUIERO <ver información sobre la aplicación> PARA <saber si es lo que estoy buscando>.

Se creará una pantalla principal a modo de presentación con una explicación de lo que es la aplicación para que el usuario no registrado pueda hacerse una idea de lo que se va a encontrar.

Registro con email

COMO <usuario no registrado> QUIERO <poder registrarme con mi correo electrónico> PARA <poder acceder a la aplicación>, CRITERIOS DE ACEPTACIÓN <validar formato de correo electrónico, la contraseña debe tener al menos ocho caracteres con números mayúsculas y minúsculas>.

Se creará una pantalla para el registro de nuevos usuarios que deberá incluir campos para nombre, email, contraseña y el aviso legal para recibir información comercial.

Validación del email

COMO <usuario registrado> QUIERO <validar mi correo> PARA <asegurarme de que el correo introducido es válido para poder recuperar mi contraseña o cerrar la cuenta>, CRITERIOS DE ACEPTACIÓN <el usuario deberá recibir un email con un enlace para confirmar el correo electrónico>.

Se creará una pantalla a la que se accederá siempre que no se tenga validado el correo electrónico. En el momento en el que se confirme la cuenta creada, esta pantalla desaparece.

Login con email

COMO <usuario registrado> QUIERO <poder iniciar sesión> PARA <acceder a mi cuenta y el resto de funcionalidades>.

Se creará una pantalla con el formulario de inicio de sesión con correo electrónico y contraseña.

6.2.4 Sprint 2

En este *Sprint* se abordan las historias de usuario correspondientes a la creación inicial del perfil de usuario. En concreto, se implementarán las siguientes historias de usuario:

Creación de perfil por primera vez

COMO <usuario registrado> QUIERO <poder completar mi perfil> PARA <aportar más información>, CRITERIOS DE ACEPTACIÓN <el tamaño máximo de la biografía es de 210 caracteres, la experiencia laboral y la formación tienen el mismo formato>.

Se creará una pantalla para que el usuario pueda completar su perfil con su experiencia laboral, estudios, descripción y foto de perfil.

Configuración de intereses

COMO <usuario registrado> QUIERO <poder incluir mis intereses y habilidades> PARA <que la plataforma pueda proponerme mejores conexiones>, CRITERIOS DE ACEPTACIÓN <el usuario podrá introducir hasta dos roles, siete cosas que busca y siete que ofrece, se deberá cubrir como mínimo una de cada, la aplicación deberá tener sugerencias predictivas para cada campo>.

Se creará una pantalla en la que el usuario pueda rellenar sus intereses. Esta información será utilizada por el algoritmo de agrupamiento para emparejar a los usuarios antes de cada reunión. También será posible saltarse esta configuración para realizarla más tarde.

Introducción de código de evento

COMO <usuario registrado> QUIERO <introducir un código de empresa o evento> PARA <acceder de manera simple a los grupos y sesiones>.

Se creará una pantalla para introducir el código de una sesión. Por el momento, para realizar las pruebas los usuarios no tendrán código de invitación y habrá que añadir su ID manualmente en base de datos.

6.2.5 Sprint 3

En este *Sprint* se consideran las historias de usuario correspondientes a la navegación y agenda de sesiones. En concreto, se implementarán las siguientes historias de usuario:

Pantalla principal

COMO <usuario registrado> QUIERO <tener una barra de navegación> PARA <poder navegar entre las distintas secciones de la aplicación>.

Se implementará una barra de navegación para que el usuario pueda moverse entre las distintas pantallas. En un principio, las diferentes pantallas estarán en blanco pero la navegación será funcional. La vista para el usuario registrado se dividirá en cinco pestañas: inicio, sesiones, contactos, notificaciones y perfil.

Agenda de sesiones

COMO <usuario registrado> QUIERO <ver todas las sesiones que están lanzadas por eventos> PARA <poder inscribirme y participar>, CRITERIOS DE ACEPTACIÓN <habrá un filtro para ordenar por fecha de comienzo o por sesiones inscritas, el calendario se divide por meses>.

Se creará una pantalla que contiene todas las sesiones que hayan publicado los grupos de interés a los que esté suscrito el usuario. En un principio, como no hay pantalla con los grupos de interés, aparecerán todas las sesiones divididas por meses, y también habrá un indicador de a cuáles ya se ha apuntado el usuario o cuáles ha descartado.

Detalles de sesión

COMO <usuario registrado> QUIERO <poder visualizar la información básica sobre las sesiones programadas> PARA <inscribirme a aquellas sesiones que me interesan>.

Pulsando en cada una de las sesiones del calendario, se mostrará una pantalla con detalles acerca de esta, incluyendo la descripción, los inscritos o las plazas máximas. También permite apuntarse a ella o descartarla.

6.2.6 Sprint 4

En este *Sprint* se tratan las historias de usuario correspondientes a las notificaciones y sala de espera de sesión. En concreto, se implementarán las siguientes historias de usuario:

Sala de espera de sesión

COMO <usuario registrado inscrito a una sesión> QUIERO <estar en una sala de espera virtual mientras no se inicia la sesión> PARA <poder entrar a tiempo>, CRITERIOS DE ACEPTACIÓN <la sala de espera durará once minutos, desde diez minutos antes del comienzo hasta el minuto de cortesía>.

Se creará una sala de espera con un contador que indique el tiempo que queda para que dé comienzo la sesión. Además, se mostrará cuántos usuarios hay ya esperando para iniciar la sesión.

Banner sala de espera

COMO <usuario registrado inscrito a una sesión> QUIERO <ver una notificación visual> PARA <poder entrar directamente en la sala de espera>, CRITERIOS DE ACEPTACIÓN <el *banner* se muestra por encima de cualquier pantalla>.

Una vez queden diez minutos para el comienzo de una sesión a la que el usuario está apuntado, se mostrará una notificación visual en forma de *banner* que el usuario podrá pulsar para entrar en la sala de espera. Como se indica en los criterios de aceptación, este *banner* debería mostrarse sobre cualquier pantalla.

6.2.7 Sprint 5

En este *Sprint* se abordan las historias de usuario correspondientes al algoritmo de agrupación y a la pantalla de valoración de una reunión. En concreto, se implementarán las siguientes historias de usuario:

Algoritmo de reparto de grupos

COMO <usuario registrado que participa en una sesión> QUIERO <ser agrupado con personas de mis intereses y con las mínimas repeticiones posibles> PARA <que las reuniones no sean monótonas y creen valor>, CRITERIOS DE ACEPTACIÓN <el máximo de personas que procesará el algoritmo es de ciento cincuenta>.

Se implementará un algoritmo de emparejamiento que consiga agrupar a todos los participantes de una sesión por intereses para permitir que un usuario participe en la ronda de reuniones de la sesión con gente afín. El algoritmo necesitará conocer los intereses especificados por cada usuario para poder así agruparlos mediante algún sistema de puntuaciones.

Feedback sesión

COMO <usuario registrado participando en una sesión> QUIERO <votar lo que me han parecido las reuniones con otros usuarios> PARA <poder generar nuevos contactos o intentar que no se me vuelva a juntar con personas con mala puntuación>, CRITERIOS DE ACEPTACIÓN <la puntuación irá de cero a cien con incrementos de diez, si ambos usuarios votan con una puntuación mayor o igual al cincuenta por ciento se compartirá la información de contacto, los nuevos contactos no se muestran hasta el final de la sesión>.

Al final de una sesión, se creará una pantalla en la que los usuarios podrán votar a los otros participantes con los que se han reunido. Además, se mostrará la foto de estos en caso de que la tengan y su nombre.

6.2.8 Sprint 6

Este *Sprint* está destinado a la parte de la aplicación correspondiente a las videollamadas. Las historias de usuario que se implementarán son:

Sesión de Speed-Networking

COMO <usuario registrado participando en una sesión> QUIERO <participar en videollamadas> PARA <conocer a los otros asistentes y así hacer contactos>, CRITERIOS DE ACEPTACIÓN <las llamadas tendrán una duración prefijada, se mostrará el tiempo restante, se podrá colgar la llamada en cualquier momento>.

En cada sesión se realizará una ronda rápida de reuniones mediante videollamada en la que cada usuario se reunirá con varios grupos de personas, siguiendo una filosofía similar al concepto de *speed dating*. Se creará la pantalla de videollamada, integrando la API de Agora con su librería para Flutter, y se podrán realizar videollamadas entre los participantes una vez el algoritmo los haya agrupado y repartido en salas.

6.2.9 Sprint 7

En este *Sprint* se aborda la parte correspondiente a la implementación de las pantallas de contactos. De esta forma, las historias de usuario que se implementarán son:

Pantalla de contactos

COMO <usuario registrado> QUIERO <tener una agenda> PARA <ver los contactos que he realizado>, CRITERIOS DE ACEPTACIÓN <por el momento sólo se mostrará su nombre, fotografía de perfil y ocupación, será necesario un buscador>.

Se creará una pantalla de contactos que muestre todos los contactos realizados con sus imágenes de perfil, nombres y trabajos actuales. En caso de no tener contactos, no se mostrará nada.

Pantalla de matches

COMO <usuario registrado que participa en una sesión> QUIERO <ver los *matches* que he realizado> PARA <poder contactar con ellos en caso de interesarme>.

Se creará una pantalla que, una vez haya terminado una sesión, mostrará los nombres de todas las personas con las que has hecho *match*. Estas personas se añadirán automáticamente a la lista de contactos.

6.2.10 Sprint 8

Este *Sprint* se centra en las historias de usuario correspondientes a la pantalla de perfil de usuario y otras mejoras visuales. En concreto, las historias de usuario que se implementarán son:

Pantalla de perfil

COMO <usuario registrado> QUIERO <poder ver y editar mi perfil> PARA <actualizar o eliminar datos>.

Se creará una pantalla con el mismo estilo que la pantalla de configuración de perfil por primera vez, que permita a los usuarios editar cualquier tipo de información sobre su cuenta o cerrar sesión.

Pantalla de sesiones vacía

COMO <usuario registrado> QUIERO <una pantalla cuando no tengo sesiones programadas> PARA <ver información de por qué no hay sesiones en mi calendario>.

Se añadirá información en el calendario de sesiones en los meses en los que no haya ninguna sesión programada para que el usuario vea una explicación.

Pantalla de contactos vacía

COMO <usuario registrado> QUIERO <una pantalla cuando no tengo contactos> PARA <ver una información de por qué no se muestran contactos>.

Se añadirá información en la pantalla de contactos cuando todavía no se haya establecido ningún contacto para que, de esa forma, el usuario vea una explicación.

6.2.11 Sprint 9

Este *Sprint* se centra en la parte correspondiente a las pantallas de inicio y notificaciones. En concreto, las historias de usuario que se implementarán son:

Pantalla de inicio

COMO <usuario registrado> QUIERO <una pantalla principal> PARA <ver grupos de interés e información sobre el evento>.

Se creará una pantalla de inicio, que se mostrará al abrir la aplicación, y que contenga un listado de los grupos de interés creados para un evento con información sobre estos.

Pantalla de notificaciones vacía

COMO <usuario registrado> QUIERO <una pantalla cuando no tengo notificaciones> PARA <saber por qué está vacía>.

Se añadirá información a la pantalla de notificaciones cuando esta esté vacía para que el usuario pueda ver una explicación.

Pantalla de notificaciones

COMO <usuario registrado> QUIERO <una pantalla de notificaciones> PARA <enterarme de los últimos eventos en mi perfil>, CRITERIOS DE ACEPTACIÓN <las notificaciones se podrán eliminar, se ordenarán por fecha>.

Se creará una pantalla de notificaciones que, en un principio, muestre notificaciones cuando tienes un nuevo contacto o cuando se ha publicado un nuevo grupo de interés. También se mostrará la fecha de recepción de la notificación.

6.2.12 Sprint 10

En este *Sprint* se abordará la implementación de la aplicación web de administración. De esta forma, las historias de usuario que se implementarán son:

Registro empresa/evento y login

COMO <administrador> QUIERO <registrarme o iniciar sesión en la aplicación web> PARA <poder administrar mi evento>, CRITERIOS DE ACEPTACIÓN <en la primera versión será necesario enviar un correo para crear el registro>.

Se creará la aplicación web comenzando con una pantalla de login/registro para que los administradores puedan acceder a los datos de su evento/empresa.

Panel de administración de grupos

COMO <administrador> QUIERO <ver todos mis grupos> PARA <poder informarme sobre ellos y consular su estado>.

La pantalla principal de la aplicación de administrador mostrará todos los grupos de interés del evento o empresa.

Dar de alta un grupo de interés

COMO <administrador> QUIERO <dar de alta un grupo de interés> PARA <poder añadir sesiones que se clasifiquen dentro de este>, CRITERIOS DE ACEPTACIÓN <los grupos se podrán guardar como borradores y no publicarlos>.

Se creará un formulario para añadir nuevos grupos a un evento, y también se podrán guardar como borradores que no se mostrarán al usuario mientras no estén publicados.

6.2.13 Sprint 11

En este *Sprint* se procede a la creación de sesiones por parte de un administrador. Las historias de usuario que se implementarán son:

Panel de sesiones en un grupo

COMO <administrador> QUIERO <ver las sesiones de un grupo> PARA <consultar información sobre ellas>.

Dentro de un grupo de interés de la aplicación de administrador, se mostrarán todas las sesiones que estén creadas dentro de este.

Alta de sesión dentro de un grupo

COMO <administrador> QUIERO <dar de alta una sesión> PARA <que los usuarios puedan participar en ella>.

Dentro de un grupo de interés de la aplicación de administrador, se podrán crear nuevas sesiones que se publicarán dentro del grupo seleccionado.

6.2.14 Planificación temporal

Una vez organizados todos los pasos del desarrollo desde el *Design Sprint* hasta que se consigue el MVP, se puede hacer una planificación con el tiempo que llevará realizar todo el proceso de desarrollo para este proyecto. Es importante recordar que la duración de cada *Sprint* en la fase de desarrollo es de una semana, lo que se traduce en 20 horas de trabajo del desarrollador. Además, cada uno de estos *Sprints* viene seguido de una sesión de revisión (y planificación del siguiente *Sprint*) de dos horas de duración. Considerando todo esto, el proyecto se ha organizado de la siguiente forma:

- Semana del 4 de mayo: *Design Sprint* y reunión del equipo.
- Semana del 11 de mayo: *Sprint* 0 y primera revisión del *Sprint* 0.
- Semana del 18 de mayo: *Sprint* 0 y segunda revisión del *Sprint* 0.
- Semana del 25 de mayo a semana del 3 de agosto (incluida): *Sprint* 1 a *Sprint* 11, con revisiones de cada *Sprint* todos los viernes.

En las figuras 6.1, 6.2, 6.3 y 6.4, se representa gráficamente el desarrollo del proyecto para los meses de mayo, junio, julio y agosto, respectivamente. Vemos como el proyecto se desarrollaría, en un caso ideal, en tres meses desde el *Design Sprint* hasta realizar los incrementos suficientes como para tener un MVP que se pueda presentar a los clientes.

6.3 Evaluación de costes

Para realizar una estimación del coste total que tendría el desarrollo de la aplicación en un entorno real, es necesario tener en cuenta los costes asociados a los sueldos de los recursos

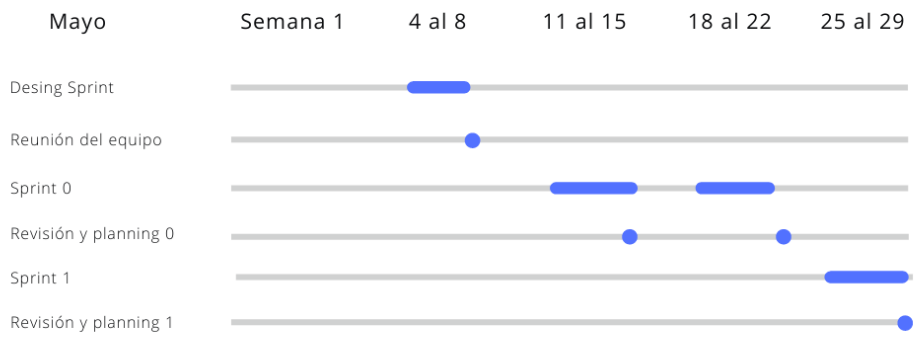

Figura 6.1: Planificación inicial del proyecto para el mes de mayo.

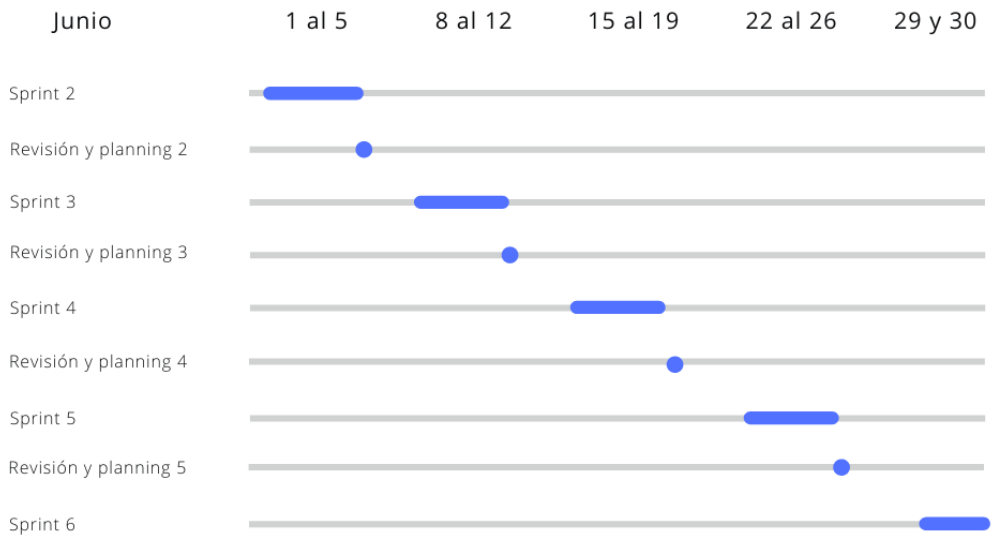

Figura 6.2: Planificación inicial del proyecto para el mes de junio.

humanos que participan en el proyecto, y los gastos correspondientes a los recursos materiales y software que se requieren en el proyecto.

En el caso del coste de los recursos humanos, vamos a continuar con la idea de que en el proyecto trabajará un solo desarrollador, el alumno. Sin embargo, para realizar una estimación lo más realista posible, vamos a simular los diferentes roles involucrados en un proceso de desarrollo de software. En este caso, tendremos un jefe de proyecto que estará representado

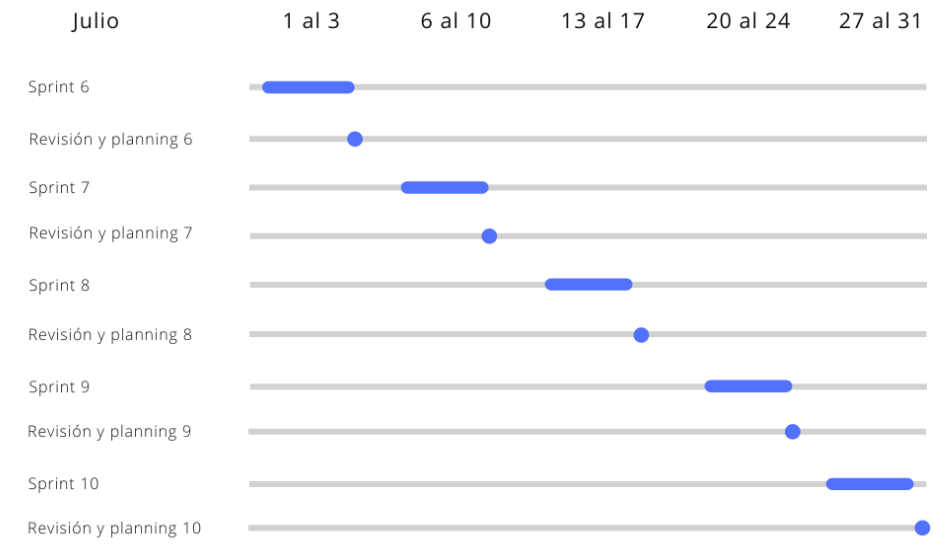

Figura 6.3: Planificación inicial del proyecto para el mes de julio.

Figura 6.4: Planificación inicial del proyecto para el mes de agosto.

por la persona de la empresa que ha supervisado en todo momento el trabajo del alumno, y los roles de analista, diseñador y desarrollador que son desempeñados por el propio alumno. Todos los trabajadores participarán en todas las reuniones a lo largo del proyecto, el diseñador se encargará también de las dos semanas del Sprint 0, y el desarrollador se encargará del resto de *Sprints*. La tabla 6.1 muestra el sueldo medio en España, según la web Ideed [19] de cada una de las personas que participarían en el proyecto:

Cargo	Sueldo medio (€/hora)
Jefe de proyecto	20.51 €
Desarrollador	16.86 €
Analista	14.15 €
Diseñador	10.26 €

Tabla 6.1: Tabla de sueldos medios por puesto de trabajo.

A modo de resumen, los procesos en los que participa cada trabajador son los siguientes:

- **Jefe de proyecto**: participa en el *Design Sprint* (6 horas), la reunión posterior (2 horas), las dos revisiones del *Sprint* 0 (4 horas) y las revisiones/planificaciones de los once *Sprints* (22 horas). Es decir, en total ha trabajado 34 horas.
- **Desarrollador**: participa en el *Design Sprint* (6 horas), la reunión posterior (2 horas), las dos revisiones del *Sprint* 0 (4 horas), todos los *Sprint* (220 horas) y las revisiones/ planificaciones de los once *Sprints* (22 horas). Por tanto, en total ha trabajado 254 horas.
- **Analista**: participa en el *Design Sprint* (6 horas), la reunión posterior (2 horas), las dos revisiones del *Sprint* 0 (4 horas) y las revisiones/planificaciones de los once *Sprints* (22 horas). Por tanto, en total ha trabajado 34 horas.
- **Diseñador**: participa en el *Design Sprint* (6 horas), la reunión posterior (2 horas), todo el *Sprint* 0 (40 horas) y las dos revisiones de este *Sprint* (4 horas), y las revisiones/planificaciones de los once *Sprints* (22 horas). Es decir, en total ha trabajado 74 horas.

Teniendo en cuenta las horas de trabajo y el sueldo medio para cada categoría, los costes totales estimados derivados de recursos humanos son de 6219 €, los cuales se desglosan con más detalle en la siguiente tabla:

Cargo	Sueldo medio (€/hora)	Horas totales	Sueldo total
Jefe de proyecto	20.51 €	34	697 €
Desarrollador	16.86 €	254	4282 €
Analista	14.15 €	34	481 €
Diseñador	10.26 €	74	759 €
			Total: 6219 €

Tabla 6.2: Tabla de sueldos totales para cada trabajador según sus horas de trabajo.

En la tabla 6.3, se muestran los costes que tendrían las licencias de software que son totalmente necesarias para el desarrollo del proyecto. Para ello, hay que tener en cuenta que los costes de Firebase y de la API de Agora dependen del uso que se le dé a la aplicación. Para completar un MVP y sus pruebas no se llegaría en ningún momento al límite gratuito que ofrecen estas herramientas, por lo que no se sumará ningún importe adicional al coste total que se está calculando. En cambio, sí que se añadirá el precio de los dispositivos hardware porque, aunque en este caso ya se disponía de todos ellos, en un proyecto real puede no ser el caso o puede que la empresa decida comprar estos equipos únicamente para este proyecto aunque se puedan reutilizar.

Elemento	Tipo	Precio aproximado
Portátil MacBook Pro	hardware	1400 €
Dispositivo Andorid de gama alta	hardware	800 €
Dispositivo iOS de útlima generación	hardware	800 €
Licencia de Sketch	software	99 €
Pago por desarrollador en Google Play	servicio	25 €
Pago por desarrollador para App Store	servicio	99 € (cuota anual)
		Total: 3223 €

Tabla 6.3: Tabla de precios de elementos de hardware, software y servicios.

El coste total estimado del desarrollo completo del MVP de la aplicación, teniendo en cuenta costes humanos y de hardware/software/servicios, es de 9442€. Sin embargo, hay que tener en cuenta que los sueldos calculados son una media del sueldo en España y pueden variar dependiendo de la región. Además, es muy posible que si la empresa ya ha realizado algún desarrollo anterior disponga de gran parte de los dispositivos de hardware o de las licencias de pago necesarias, por lo que esto también podría reducir el coste.

En resumen, las horas de trabajo totales y el coste estimado de personal y software/hardware es el siguiente:

- **Recursos humanos:** 396 horas totales con un coste de 6219 €.
- **Recursos de hardware/software:** 3 dispositivos hardware y 3 servicios con un coste aproximado de 3223 €.
- **Coste total:** 9442 €.

6.4 Revisión final de la planificación temporal

No se entrará demasiado en detalle en este apartado, pues la planificación final fue muy similar a la planificación realizada inicialmente que ya se ha explicado. Como la aplicación se ha desarrollado de forma modular y cada semana se entregaba un producto funcional, no ha sido necesario modificar ningún *Sprint*, ya que cada semana se añadía un módulo nuevo que apenas se relacionaba con el anterior y los resultados de una semana no afectaban a la siguiente. El único contratiempo que se ha tenido ha sido un *spike* introducido en el *Sprint* 5, que alargó este una semana más, lo que afectó a la duración del proyecto pero sólo en siete días más (se explicará porque ocurrió en el capítulo de implementación 8). De esta forma, el proyecto estaba planificado que finalizara el día siete de agosto pero, en realidad, finalizó el día catorce, un retraso que ojalá se produjera en todos los procesos de desarrollo pues este suele ser mucho mayor.

Capítulo 7

Diseño

En este capítulo se explican las diferentes decisiones de diseño tomadas, tanto para la arquitectura interna de la aplicación como para el diseño de la base de datos y de la interfaz de usuario.

7.1 Diseño de la base de datos

La base de datos que se aloja en Firebase es una base de datos NoSQL, por lo que los datos no se almacenan en tablas, sino que se almacenan como elementos JavaScript Object Notation (JSON). Por esta razón, la base de datos de la aplicación se divide en dos grandes entidades: usuarios y eventos. También se añaden dos pequeñas entidades que se utilizan para guardar información sobre los intereses que han cubierto los usuarios, como se explicará en las siguientes secciones.

7.1.1 Usuarios

La entidad usuarios contiene la información de todos los usuarios registrados en la plataforma, sin diferenciar si son administradores o usurarios de la aplicación. Firebase asigna automáticamente un ID de usuario cada vez que se registra un nuevo usuario y se guarda en la base de datos. Como no se diferencia entre administradores y usuarios, en la base de datos se añade un campo "isAdmin", con valor de tipo cadena de texto, que indica que evento gestiona. De esta forma, una persona que sea administrador de un evento podrá igualmente utilizar esa misma cuenta para hacer un uso normal de la aplicación. El resto de campos que se guardan para un usuario son los siguientes:

- **Nombre (name):** tipo String, nombre introducido durante el registro.
- **Apellidos (surname):** tipo String, apellido/s introducidos durante el registro.

- **Estado de configuración de intereses (topicsConfigured):** tipo String que representa el estado en que se encuentra la configuración de intereses. Este estado puede ser *False* (sin configurar), *Skipped* (omitida) o *True* (configurada). Se trata de un campo útil para conocer si los usuarios se interesan por la configuración de intereses y para poder avisar en algún momento de que esta configuración está sin cubrir.

- **Estado de configuración de perfil (userDataConfigured):** igual que el campo anterior pero, en este caso, se utiliza para definir el estado de la configuración de experiencia laboral, estudios, descripción y foto de perfil.

- **Activación de marketing (marketingEnabled):** tipo booleano, indica si el usuario ha marcado la casilla para recibir información comercial.

- **Descripción de perfil (info):** tipo String, guarda la descripción del perfil de usuario.

- **URL de la imagen de perfil (imageUrl):** tipo String que contine la URL de la imagen de perfil que se encuentra en Firebase Storage. La utilidad de este campo se explica con más detalle en la siguiente sección.

- **Contactos (contacts):** tipo lista de Strings, contiene los IDs de los contactos realizados.

- **Sesiones descartadas (discardedSessions):** tipo lista de Strings, contiene los IDs de las sesiones de las que se ha descartado su participación.

- **Sesiones inscritas (sessions):** tipo lista de Strings, contiene los IDs de las sesiones en las que se ha inscrito.

- **Grupos suscritos (groups):** tipo lista de Strings, contiene los IDs de los grupos en los que está suscrito.

- **Notificaciones (notifications):** de tipo colección, contiene las notificaciones que se muestran al usuario. Se explica con más detalle a continuación.

- **Trabajos (jobs):** de tipo mapa, guarda los trabajos configurados por el usuario en su perfil. Se explica a continuación.

- **Estudios (studies):** de tipo mapa, guarda los estudios configurados por el usuario en su perfil. Se explica a continuación.

- **Intereses (topics):** de tipo mapa, guarda los intereses configurados en el perfil. Se explica a continuación.

- **Votos (lastRooms):** de tipo mapa, guarda la puntuación que el usuario ha otorgado a otros usuarios en sesiones anteriores. De esta forma, el algoritmo de agrupamiento podrá tener en cuenta estas puntuaciones para nuevas sesiones en las que participen las mismas personas.

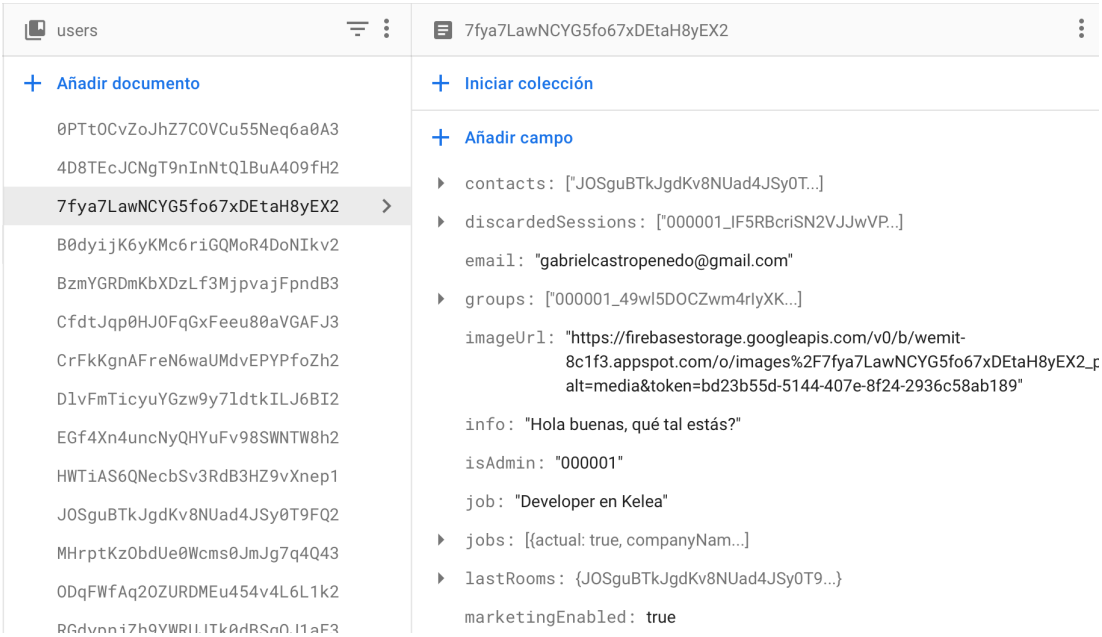

Figura 7.1: Base de datos de usuarios

A continuación, vamos a explicar con más detalle algunos de los elementos que forman parte de la entidad de usuarios:

URL de la imagen de perfil con Firebase Storage

Como ya se explicó en el capítulo de fundamentos tecnológicos 3, Storage es un servicio de Firebase que permite almacenar archivos en la nube. Para el caso de almacenar las imágenes de perfil de los usuarios, era necesario hacerlo en la nube puesto que los contactos que haga el usuario también tienen la posibilidad de visualizar su imagen de perfil, por lo que no tiene sentido almacenar ningún archivo de forma local. Aprovechando que Firebase ofrece este servicio, cada vez que un usuario añade o cambia su imagen de perfil, se crea en Storage una entrada con el archivo correspondiente a la imagen, y el nombre siempre será el ID de usuario de la persona que suba la imagen. De esta forma, siempre se sobrescribe la imagen anterior en caso de que un usuario cambie su foto de perfil.

En la figura 7.2, se puede ver un ejemplo del uso de Firebase Storage para almacenar imágenes de un usuario.

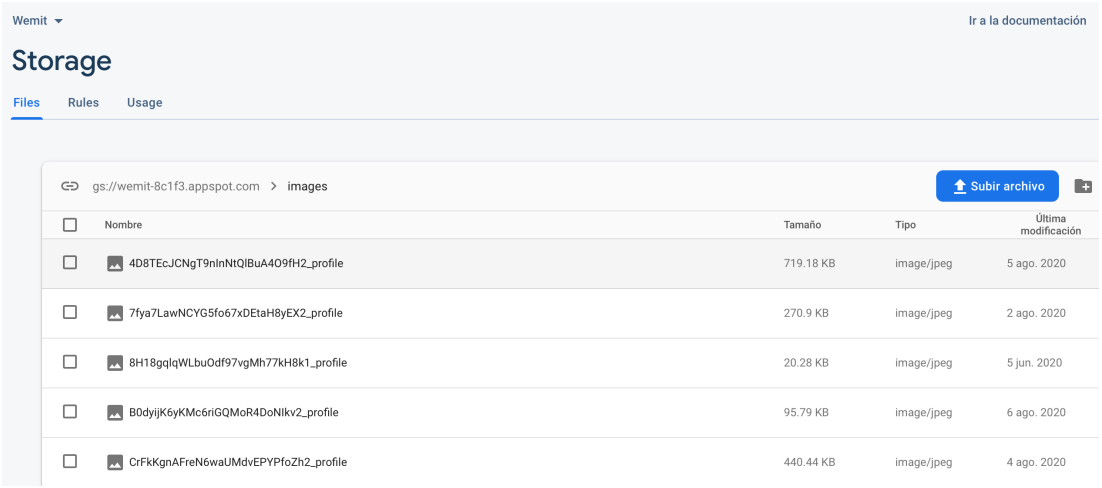

Figura 7.2: Imágenes de usuario en Firebase Storage

Entidades de tipo colección

Dentro de la base de datos de cada usuario, se usa una entidad de tipo colección para almacenar sus notificaciones. Una entidad de tipo colección se puede ver como una nueva entidad dentro del propio usuario que está compuesta de nuevos elementos con sus IDs únicos y sus correspondientes campos para de cada elemento. Se ha decidido guardar las notificaciones de esta forma porque la idea es que, en algún momento del ciclo de vida de la aplicación, el usuario reciba e interactúe en gran medida con ellas, por lo que es interesante que estas tengan un ID que las identifique para poder realizar operaciones de insertado, edición y borrado de forma más cómoda. También puede ocurrir que esta colección crezca mucho por lo que insertarla en forma de mapa sería menos eficiente. En caso de querer limitar el alcance de una consulta a la base de datos, por ejemplo para coger las últimas diez notificaciones, no se podría hacer si las hemos insertado como un mapa. Teniendo en cuenta esto, cada notificación se divide en los siguientes campos:

- **Nombre (name):** de tipo String, nombre principal que muestra la notificación.
- **URL de la imagen principal (imageUrl):** de tipo String, representa la URL de la imagen de perfil de la persona que "envía" la notificación. Puede ser un usuario o un evento.
- **Texto de información (name):** de tipo String, corresponde al texto que completa la información de una notificación, como puede ser el nombre de un nuevo grupo creado para un evento (opcional).

- **URL de la imagen de información (infoImageUrl):** de tipo String, representa la URL de una imagen que se quiera incluir en la notificación, como puede ser la imagen de un nuevo grupo de interés (opcional).
- **Tipo (type):** de tipo String, indica el tipo de notificación y resulta útil por si en algún momento se quiere filtrar en la pantalla de notificaciones o cambiar el estilo de cada una por tipos.
- **Fecha (time):** de tipo TimeStamp, se corresponde con la hora en la que se generó la notificación, y resulta útil para poder ordenarlas o eliminar las más antiguas.

Trabajos

Este campo de la entidad usuarios se inserta en la base de datos como un mapa. Un mapa se puede ver como una lista de elementos pero con un identificador para cada uno. En este caso, los elementos no se insertan en una colección porque no se van a hacer consultas ordenadas sobre ellos y, además, esta información no se edita con frecuencia. En particular, el mapa de trabajos está formado por los siguientes campos:

- **Actual (actual):** de tipo booleano, indica si es el trabajo que se está desarrollando actualmente.
- **Nombre de la compañía (companyName):** de tipo String, nombre de la empresa.
- **Fecha de comienzo (startDate):** de tipo TimeStamp, fecha de comienzo de este trabajo.
- **Fecha de finalización (endDate):** de tipo TimeStamp, indica la fecha en la que se dejó de realizar el trabajo. En caso de ser el trabajo actual, el campo será nulo.
- **Rol (role):** de tipo String, indica el puesto desempeñado.

Estudios

Al igual que el mapa de trabajos, estos elementos se insertan en la base de datos como un mapa con los siguientes campos:

- **Actual (actual):** de tipo booleano, indica si es el estudio que se está cursando actualmente.
- **Nombre de la institución (school):** de tipo String, nombre de la institución.
- **Fecha de comienzo (startDate):** de tipo TimeStamp, fecha de comienzo de esta formación.

- **Fecha de finalización (endDate):** de tipo TimeStamp, fecha de finalización de la formación. En caso de ser la formación actual, el campo será nulo.
- **Grado (grade):** de tipo String, identifica el estudio realizado.

Intereses

Este campo se puede editar con algo más de frecuencia que los anteriores, pero el número de elementos es muy pequeño y no causa problemas tenerlo insertado como un mapa. Está formado por los siguientes campos:

- **Roles (roles):** de tipo lista de Strings, guarda los roles que desempaña el usuario actualmente como, por ejemplo, desarrollador y analista.
- **Ofrecimientos (give):** de tipo lista de Strings, indica los conocimientos que puede ofrecer el usuario a otras personas interesadas como, por ejemplo, Flutter.
- **Búsquedas (receive):** de tipo lista de Strings, indica los conocimientos que busca la persona en otros usuarios como, por ejemplo, Scrum.

7.1.2 Eventos

La entidad eventos contiene la información sobre todos los eventos/empresas que organizan sesiones. Cada evento se identifica con un ID único y posee una serie de campos entre los que destacan dos colecciones: grupos y sesiones. En concreto, los campos correspondientes a una entidad evento son los siguientes:

- **Nombre (name):** tipo String, indica el nombre del evento.
- **Descripción (description):** tipo String, contiene la descripción del evento.
- **Participantes (participants):** de tipo lista de Strings, guarda los IDs de los usuarios invitados al evento.
- **Grupos de interés (groups):** de tipo colección, se explica a continuación.
- **Sesiones (sessions):** de tipo colección, se explica a continuación.
- **Estado de las sesiones (sessionState):** de tipo mapa, representa el estado en el que se encuentran las sesiones que han comenzado para el evento. Este estado puede ser *calculatingRooms* (calculando las salas), *talking* (en reunión), *finshed* (terminada) o *noParticipants* (terminada porque no hay suficientes usuarios conectados).
- **Notificaciones (notifications):** tipo lista de Strings, guarda los IDs de las sesiones de las cuales se tiene que mostrar la notificación (*banner*) para poder entrar.

Grupos de interés

Contiene la información de los grupos de interés creados para un determinado evento con su correspondiente identificador, y sus campos son los siguientes:

- **Nombre (name):** de tipo String, nombre del grupo.
- **Descripción (description):** de tipo String. descripción del grupo.
- **URL de la imagen (imageUrl):** de tipo String, contiene la URL de la imagen del grupo.
- **Borrador (isDraft):** de tipo booleano, indica si es un borrador y por lo tanto no debe mostrarse.
- **Intereses (topics):** de tipo lista de Strings, representa la lista de intereses para que los usuarios vean si les puede encajar el grupo.

Sesiones

Contiene la información de las sesiones de un evento con su identificador, y sus campos son los siguientes:

- **Nombre (name):** de tipo String, título de la sesión.
- **Duración (duration):** de tipo String, indica la duración aproximada de la sesión.
- **Descripción (description):** de tipo String, descripción del tema de la sesión.
- **ID de grupo (groupId):** de tipo String, contiene el ID del grupo de interés al que pertenece la sesión.
- **Máximo de participantes (maxParticipants):** de tipo Number, indica el número máximo de participantes.
- **Participantes (participants):** de tipo lista de Strings, contiene los IDs de los participantes que se han inscrito en la sesión.
- **Personas por sala (personsOnRoom):** de tipo Number, indica el número de personas en cada sala (videollamada).
- **Duración de sala (roomDuration):** de tipo Number, indica la duración en minutos de cada sala, es decir, el tiempo que durará cada reunión o videollamada.
- **Rondas (rounds):** de tipo Number, indica el número de rondas de *speed networking* que se realizarán en esa sesión.

- **Fecha de inicio (startDate):** de tipo TimeStamp, indica la fecha de inicio de la sesión.
- **Tipo (type):** de tipo String, útil por si en algún momento se implementan reuniones que funcionen de otra forma, por el momento su valor es *SpeedNetworking*.
- **Siguientes salas (nextRooms):** de tipo mapa, se utiliza para guardar en base de datos los resultados del algoritmo de agrupación. Contiene un conjunto de salas formadas por los IDs de los participantes que entrarán en cada una de ellas en la siguiente ronda. Además, este campo se actualiza entre ronda y ronda usando el algoritmo de agrupamiento.

7.1.3 Entidades para intereses

Para guardar información sobre los intereses mostados por los usuarios se utilizan dos entidades adicionales: *topics* y *defaultTopics*. Estas entidades se insertan en la base de datos para mejorar la experiencia cuando los usuarios detallan sus intereses, ya que es fundamental una buena configuración de estos para que el algoritmo haga un emparejamiento de forma correcta. Estas entidades están formadas por roles, búsquedas y ofrecimientos al igual que el campo intereses de tipo mapa de la entidad usuario. En *defaultTopics* se guardan los intereses que aparecerán por defecto antes de que el usuario realice la configuración con el objetivo de que tenga un ejemplo de a qué se refiere cada uno. Una vez el usuario configure sus intereses y los guarde, estos se insertarán en la entidad *topics* para que puedan ser utilizados como sugerencia para el siguiente usuario que los vaya a configurar.

Esta decisión de diseño se ha tomado para evitar que pequeñas diferencias a la hora de escribir los intereses provoquen que el algoritmo no junte a dos personas con los mismos intereses pero que han utilizado una descripción diferente para ellos. Por ejemplo, una persona puede escribir que busca gente que tenga conocimientos de Agile y otra puede escribir que ofrece conocimientos sobre metodologías ágiles. Con esta solución, cuando la segunda persona configure sus intereses le saldrá Agile como sugerencia cuando esté escribiendo “ágiles” y podrá seleccionarla directamente. De esta forma, se evita tener usuarios con intereses iguales que no se escriban igual y que el algoritmo no los pueda detectar. Cuantos más usuarios se hayan registrado, mayor será la cantidad de sugerencias que pueden aparecer.

7.2 Base de datos en tiempo real

Otra de las bases de datos que ofrece Firebase es Realtime Database. Se trata también de una base de datos NoSQL pero optimizada para su uso en aplicaciones que necesitan accesos en tiempo real. En la aplicación desarrollada, su uso permitirá llevar un registro de los usuarios

que están conectados a una sesión. Este registro sirve para controlar qué usuarios están listos para entrar en la siguiente ronda o qué usuarios se han desconectado en medio de una sesión y, por tanto, el algoritmo no los tendrá que tener en cuenta para la siguiente ronda. Otra de sus funciones es llevar un control de los votos de los usuarios. Por tanto, en esta base de datos se van a guardar los votos que se han ido realizando durante una sesión para que, cuando esta termine, se puedan calcular los nuevos contactos. La figura 7.3 muestra como se inserta el ID de un usuario que se ha conectado a la sala de espera de una sesión.

Figura 7.3: Usuario conectado en Realtime Database

7.3 Arquitectura de la aplicación móvil

Tanto la aplicación móvil como la aplicación de administración partirán del mismo patrón de diseño para su implementación aunque, únicamente, se explicará cómo se diseñó la aplicación móvil pues esta tiene muchas más funcionalidades y la aplicación de administración, que por el momento es mucho más sencilla, sigue una filosofía de diseño similar. El patrón de diseño utilizado es una modificación del ya conocido MVC o Modelo-Vista-Controlador, llamado *Scoped Model*.

7.3.1 Scoped Model

Scoped Model es un patrón de diseño pensado para el desarrollo de aplicaciones en Flutter que se utiliza gracias a un paquete que implementa sus funcionalidades y que es necesario importar en el proyecto de Flutter. Se puede considerar una modificación del patrón MVC que junta en un único elemento el modelo y el controlador por lo que, básicamente, tendremos archivos que implementan modelos, y archivos que implementan vistas. Para tener una arquitectura correcta, las vistas únicamente deben implementar los apartados visuales de la

aplicación, mientras que el modelo debe ser el encargado de gestionar los estados y la ejecución de las funciones.

El paquete de *Scoped Model* para Flutter proporciona los métodos para la implementación de este patrón de diseño. En este caso, tenemos que insertar los elementos visuales de Flutter dentro de un elemento *ScopedModelDescendant* e indicarle cuál es el modelo del que desciende. El paquete se encarga de que, cuando haya un cambio en el modelo (cambios en variables o ejecución de funciones), la vista se reconstruya para mostrar los nuevos datos, por lo que la vista sólo necesita acceso a las variables y en ella no se implementará ningún tipo de lógica de negocio.

Las aplicaciones basadas en *Scoped Model* se pueden diseñar de dos formas: 1) con un modelo principal del que heredan otros sub-modelos más sencillos, 2) con un modelo por cada funcionalidad o incluso vista. Para esta aplicación, se ha optado por el segundo método, creando un modelo por cada funcionalidad principal de la aplicación, teniendo así un modelo para usuario, otro para sesiones, otro para notificaciones, etc. Si alguno de los modelos necesita datos de otros, por ejemplo, si el modelo de notificaciones necesita el ID de usuario, este se le pasará como parámetro en el momento de inicializarlo. De todas formas, la primera de las opciones de diseño también habría sido buena, pues se puede dividir el modelo principal en módulos que realizarían la misma función.

En conclusión, este tipo de arquitectura nos proporciona una gran modularidad pudiendo añadir un número infinito de nuevos modelos para nuevas funcionalidades y una abstracción de la vista sobre la lógica de negocio, lo que lo hace un patrón muy útil y bien diseñado.

7.3.2 Navegación y tiempos de carga dentro de la aplicación

Uno de los requisitos para el desarrollo era que la aplicación minimizara, en la medida de lo posible, los tiempos de carga. Por esa razón, la aplicación se ha diseñado de forma que la vista principal, que contiene la navegación, se encargue de invocar todas las funciones de carga de datos que puedan ralentizar su uso y de gestionar la navegación entre pantallas. El hecho de que esta vista principal se muestre sobre las demás también nos proporciona la ventaja de que, para mostrar cualquier elemento prioritario por encima de otras pantallas, simplemente habría que incluirlo en esta vista. Un ejemplo de esto sería el caso del *banner* que se muestra cuando se abre la sala de espera de una sesión y que debe permanecer en todas las vistas. La imagen 7.4 muestra una aproximación de cómo la vista principal maneja el resto de vistas.

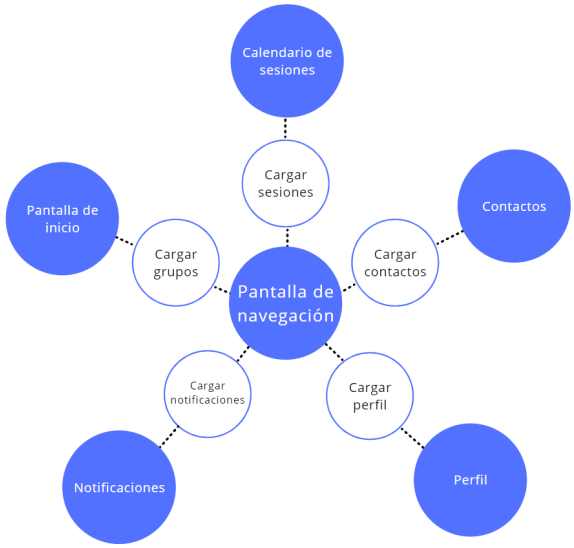

Figura 7.4: Esquema de funcionamiento de navegación y carga de datos.

7.3.3 Entidades persistentes

La aplicación contiene una gran cantidad de clases persistentes, básicamente una para cada elemento compuesto de la base de datos. Estas clases son necesarias para pasar los objetos de la base de datos a objetos que comprenda la aplicación y viceversa. Por tanto, se van a considerar las siguientes clases: **user, topic, job, education, group, event, session, call, room, partner, contact y notification**. No se entrará en más detalle porque sus atributos son los mismos que se han explicado para los elementos correspondientes de la base de datos.

7.4 Diseño de la interfaz de usuario

En esta sección se mostrará el resultado de las dos semanas dedicadas al *Sprint* 0, en el que se hizo un diseño, lo más realista posible, de la interfaz de la aplicación para hacerse una idea de como sería el producto final. En la reunión que se realizó una vez acabado el *Design Sprint*, se decidieron los colores principales, además del nombre y el logo de la aplicación.

7.4.1 Decisiones de diseño

Durante las reuniones que se realizaron para revisar el avance en el diseño de la aplicación, se tomaron unas decisiones sobre cómo tendría que ser la interfaz de la aplicación:

- El nombre de la aplicación sería wemit. Este nombre surge de *we* en inglés "nosotros" y *meet* en inglés "reunirse" o "conocerse", que representa bien el objetivo de la aplicación.
- El diseño de la aplicación tiene que estar muy cuidado para que dé la sensación de haber sido desarrollada con un objetivo profesional y haber contando con un buen equipo de diseño, aunque no haya sido el caso.
- El diseño de la aplicación móvil y web seguirán el mismo patrón de diseño y esquema de colores para que la sensación de integración y de que pertenecen a la misma marca sea mayor.
- Se intentará, en la medida de lo posible, que las diferencias en la navegación y comportamientos de los distintos sistemas operativos no afecten al diseño de la aplicación y se vea exactamente igual en cualquier dispositivo.

A continuación, se muestran las imágenes del diseño inicial realizado con Sketch. Es importante recalcar que el diseño del MVP no será exactamente igual pues no cuenta con absolutamente todas las funcionalidades analizadas.El diseño de la aplicación web se puede consultar de ser necesario en el anexo A, en general este sigue las mismas directrices.

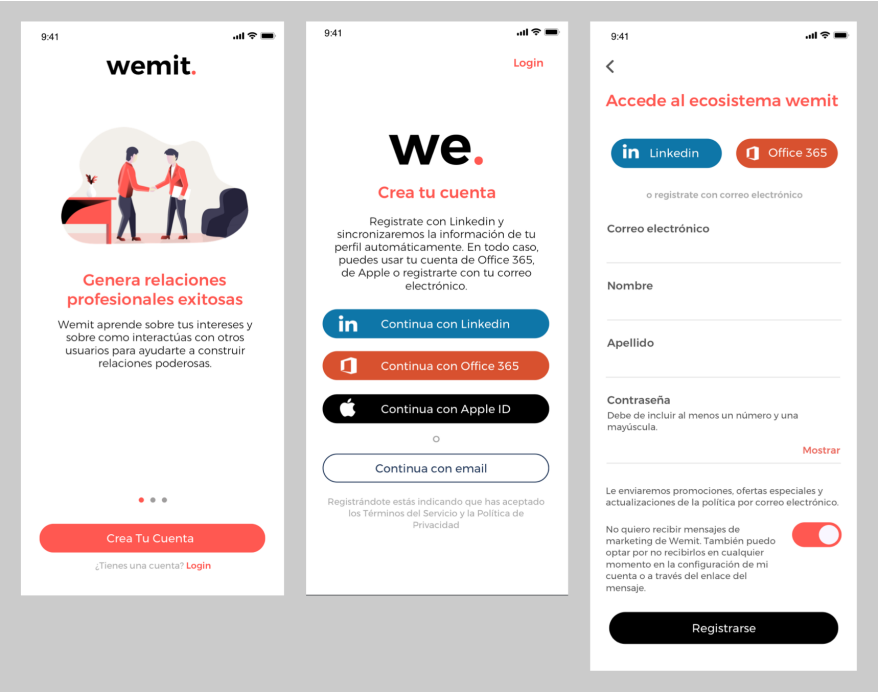

Figura 7.5: Pantallas de bienvenida y registro.

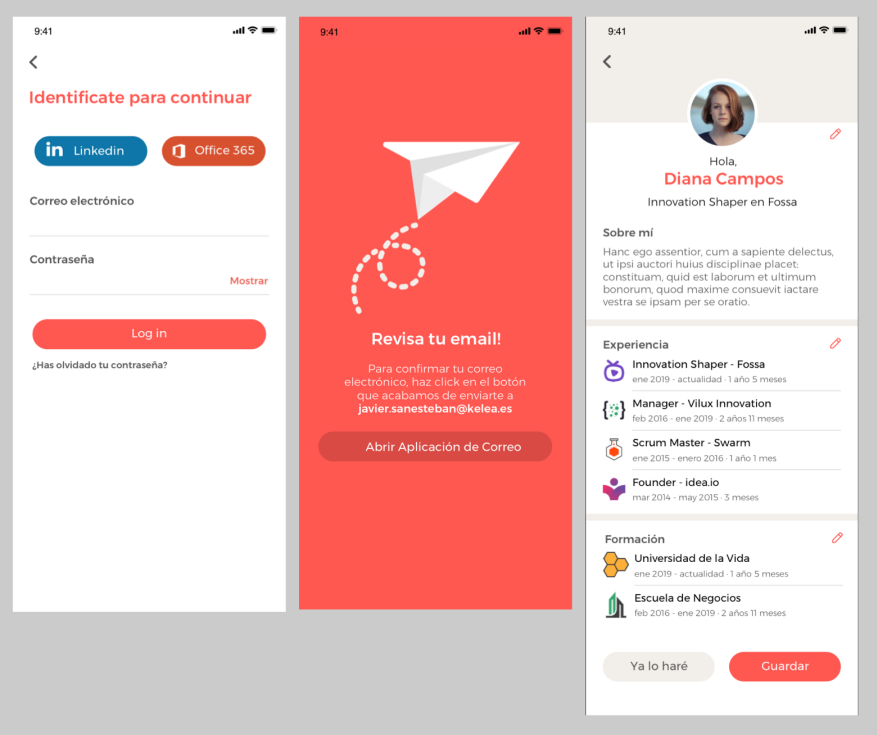

Figura 7.6: Pantallas de login, validación email y perfil.

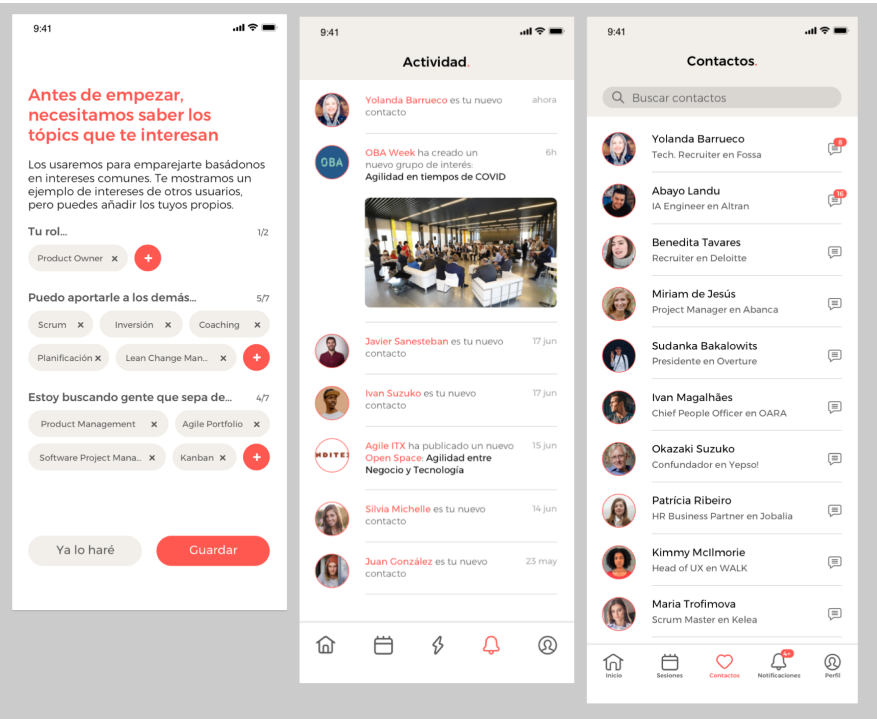

Figura 7.7: Pantallas de configuración de intereses, notificaciones y contactos.

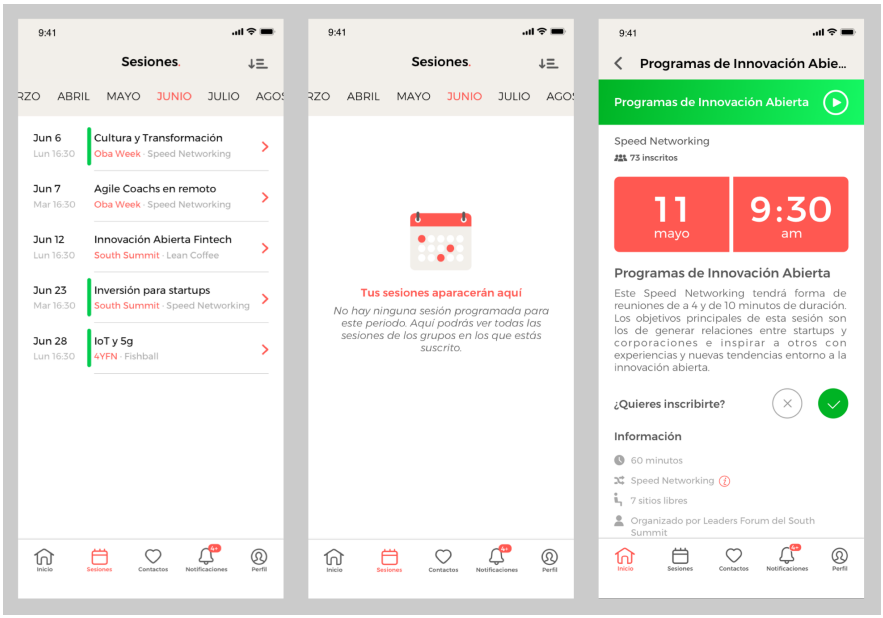

Figura 7.8: Pantallas de calendario y detalles de sesión.

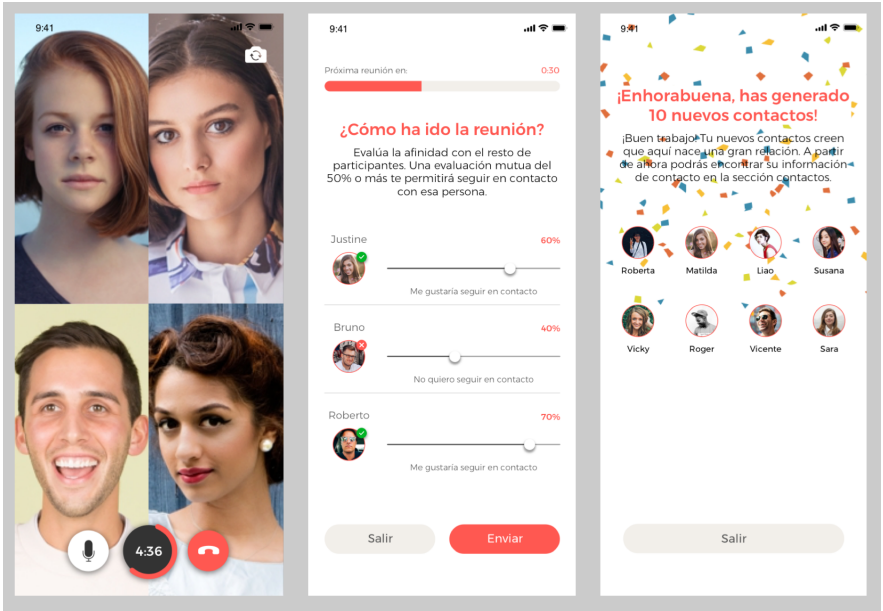

Figura 7.9: Pantallas de llamada, valoración y *matches*.

Capítulo 8

Implementación

En este capítulo se explicarán principalmente los datos y decisiones relevantes del proceso de implementación, tanto en la parte de *frontend* como en la parte de *backend*. Comentar todos los detalles de implementación de la aplicación resulta imposible, así que nos centraremos en algunas de las características más destacadas.

8.1 Algoritmo de agrupación de usuarios

Una de las partes más importantes y complejas implementadas en esta aplicación es el algoritmo que permite agrupar a los participantes de las sesiones por intereses comunes. Es esencial que este algoritmo funcione correctamente para que los usuarios tengan las sensación de que las personas con las que se reúnen son afines a sus gustos ya que, si esto no es así, parecerá que las reuniones se organizan de forma aleatoria y la aplicación perdería uno de sus principales atractivos. A continuación, se explican las diferentes fases que se llevaron a cabo para su desarrollo.

8.1.1 Investigación

Durante la planificación inicial del proyecto no se planteó la idea de llevar a cabo una investigación exhaustiva del algoritmo. En un principio, se sabía lo que se quería conseguir (repartir a los participantes en grupos por intereses durante varias rondas y conseguir que no se repitieran las reuniones), pero no cómo llegar a ello. Debido a esto, hubo que introducir una pausa en el proyecto o *spike*, como se explicó en el capítulo de metodología 4. Esta pausa afectó al *Sprint* número 5, por lo que la duración de este pasó de ser de una semana a dos semanas.

La primera aproximación que se planteó para implementar el algoritmo fue utilizar métodos basados en teoría combinatoria para generar los grupos. Sin embargo, esta aproximación únicamente buscaba soluciones que agrupaban a los participantes en varias rondas sin

repetición y, además, era muy costosa computacionalmente. En la práctica se observó que, dependiendo del número de participantes y rondas, el tiempo de cálculo aumentaba exponencialmente. Este problema se podría solucionar dividiendo el grupo inicial de participantes en subgrupos y haciendo los cálculos sobre estos subgrupos, pero esto hacía más complicado garantizar la no-repetición en sesiones con muchas rondas. Por último, esta aproximación aún no tenía en cuenta los gustos de los participantes, por lo que añadir esa variable supondría aumentar aún más el tiempo de computación y no lo haría viable, por lo que se descartó la idea.

Tras un par de días de investigación, surgió la idea de implementar un algoritmo genético [20]. En este caso, su propósito sería generar grupos maximizando una puntuación que representa la afinidad entre las personas de cada grupo. Un algoritmo genético consiste, como la selección natural de Darwin, en ir evolucionando una población de individuos de forma que los más "aptos" se reproduzcan y los menos "aptos" mueran para conseguir, de esa forma, terminar con una población formada por los individuos más "aptos". En este caso, el término individuo hace referencia a una posible agrupación de participantes, mientras que el término "apto" hace referencia a aquellos individuos de la población que presentan una mejor puntuación de acuerdo a la métrica establecida. Además se introduce el factor de la mutación, mediante el cual algunos individuos "cambian" sus características de modo aleatorio, lo que puede generar un individuo con mejores o peores características. En cualquier caso, este mecanismo permite aumentar la diversidad de la población. Un algoritmo genético era ideal para el tipo de problema que se quería resolver, pues siempre proporcionará una solución en el periodo de tiempo que le marquemos, aunque sea más o menos óptima.

8.1.2 Implementación inicial

Una vez investigadas las diferentes posibilidades, era la hora de implementar la solución que parecía que más se ajustaba a las necesidades de la aplicación.

Para comenzar, la población inicial se formó creando grupos aleatorios compuestos por los participantes, indicando antes el número de personas que tendría que haber para cada sala. Para que el agrupamiento inicial fuese correcto, las personas se meterían en los grupos de forma que estos quedaran lo más equilibrados posible en número. Por ejemplo, si el algoritmo recibe seis participantes y quiere grupos de cuatro personas, no se establecerá un grupo de cuatro y uno de dos, sino que se realizarán dos grupos de tres personas. El número de individuos que hay en la población inicial se puede ajustar con un parámetro, ahora mismo se utilizan veinte, por lo que hay veinte reuniones distintas posibles.

El siguiente paso es tener una función que calcule la afinidad de los grupos y de cada individuo de la población. Como cada usuario tiene tres tipos de intereses (rol, busca y ofrece), se ha asignado una determinada puntuación a cada tipo de interés de forma que, si dos usuarios coinciden, se suma esta cantidad. El comportamiento de esta función para dos personas X e Y es el siguiente:

1. Puntuación inicial = 0.
2. Para cada rol de X, si coincide con un rol de Y se suma a la puntuación acumulada 0.5.
3. Para cada busca de X, si coincide con un ofrece de Y se suma a la puntuación 5.
4. Para cada ofrece de X, si coincide con un busca de Y se suma a la puntuación 3.
5. Si ya se han reunido, y la puntuación de X a Y es mayor o igual que 70% se resta 5.
6. Si ya se han reunido, y la puntuación de X a Y está entre el 31% y el 69% se resta 10.
7. Si ya se han reunido, y la puntuación de X a Y es menor o igual que 30% se resta 15.

Vemos como también se resta cierta puntuación en caso de que las dos personas ya se hayan reunido anteriormente, y la cantidad que se resta depende de la puntuación que se hayan dado. Esto representa un factor de penalización en la función de evaluación para minimizar la posibilidad de que se repitan grupos con los mismos individuos para diferentes reuniones. Estos cálculos se realizan para cada uno de los participantes dentro de cada grupo con todo el resto del grupo, y todas las puntuaciones sumadas determinan la puntuación total de ese grupo. De esa forma, sumando todas las puntuaciones de los grupos, obtenemos la puntuación de un individuo de la población.

Una vez se define la función de evaluación para calcular lo "bueno" que es un individuo (agrupación), comienza realmente la ejecución del algoritmo iterativo. En el caso de la aplicación, hemos elegido que se hagan mil iteraciones puesto que se ha observado que, alrededor de este número, se empiezan a obtener buenos resultados. De todas formas, el número de iteraciones del algoritmo es otro parámetro de diseño.

Los pasos que realiza el algoritmo en cada iteración son los siguientes:

1. Calcular la puntuación de cada individuo de la población.
2. Ordenar la población de puntuación más alta a más baja.
3. Eliminar de la población los peores individuos.

4. Insertar en la población mutaciones de los mejores individuos.

5. Repetir.

Mediante este proceso y a medida que van avanzando las iteraciones, conseguimos que la población vaya mejorando pues los individuos más aptos siempre sobreviven, y las mutaciones pueden resultar en individuos mejores que sus "padres", por lo que los acaban sustituyendo. Con respecto a la formulación clásica de los algoritmos genéticos, el algoritmo propuesto se salta el paso de cruzar individuos ya que este tipo de operaciones puede dar lugar a un conjunto de grupos que contenga personas duplicadas y, por lo tanto, no se conseguirían buenos resultados.

8.1.3 Primera prueba

Para realizar una prueba del algoritmo, se generó una población de ciento cincuenta personas con intereses aleatorios sacados de una lista escrita a mano. Se hicieron pruebas para generar salas de dos, tres y cuatro personas. Los resultados del agrupamiento con el algoritmo genético fueron en general buenos (las salas estaban formadas por personas con coincidencias), lo que nos confirmaba que el algoritmo funcionaba. Sin embargo, esta primera aproximación planteaba un posible problema en la práctica. Aunque el algoritmo no estaba desplegado en Firebase y no conocemos a priori el poder computacional de las máquinas de Google, en el ordenador personal del desarrollador la función tardaba en ejecutarse entre cuarenta y cincuenta segundos. No parece demasiado tiempo para ser cerca de doscientas personas, y puede que este tiempo se mejore una vez desplegado, pero en una aplicación que necesita realizar este agrupamiento varias veces por cada sesión, los tiempos de espera se sumarían y llegarían a ser un tema que perjudicaría a los usuarios. Esto incitó a intentar realizar mejoras.

8.1.4 Mejoras

Para reducir el tiempo de ejecución, lo primero que se hizo fue intentar bajar el número de iteraciones del algoritmo, pero los resultados que arrojaba eran peores y no era un cambio factible. La siguiente idea fue analizar la duración de cada proceso dentro de una iteración, para ver si alguno era el culpable de la tardanza. Con este análisis se descubrió que el problema estaba en que, cada vez que se ordenaba la población (paso 2), se ejecutaba la función de evaluación para calcular la puntuación de cada uno de los individuos. Por esa razón, se decidió guardar en una variable las puntuaciones ya calculadas de forma que, si un individuo o sus componentes no cambiaban (sobrevive), no era necesario ejecutar la función de puntuación. A esto se añade que, cada vez que se ordena la población de mejor a peor, ya no es necesario llamar otra vez a la función de puntuación, por lo que es mucho menos costoso.

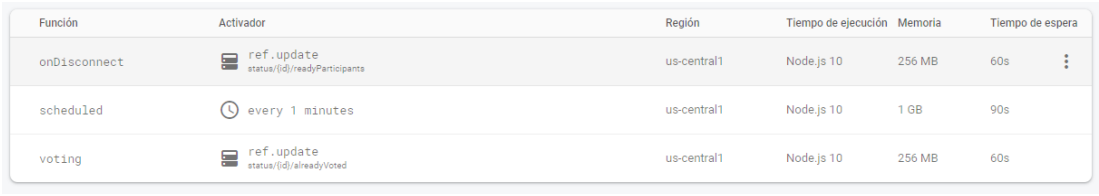

Figura 8.1: Funciones desplegadas en Firebase Functions.

8.1.5 Prueba final

Una vez introducidos los cambios comentados en la sección anterior, se realizó la misma prueba que en la primera ocasión. Los resultados del agrupamiento no habían cambiado, pero la mejora en el rendimiento del algoritmo fue sorprendente ya que, únicamente, tardaba entre cuatro o cinco segundos en realizar el cálculo. Es decir, el simple cambio de guardar las puntuaciones multiplicó la velocidad por diez. En este punto, el algoritmo ya estaba listo para ser desplegado.

8.2 Firebase Functions

Firebase Functions es el servicio de Firebase que permite alojar las funciones del *backend* de la aplicación. Estas funciones se ejecutan en una máquina que aprovisiona Google por lo que su carga computacional no recae en la aplicación. Se pueden definir funciones que se invocan a través de peticiones HyperText Transfer Protocol (HTTP), a través de eventos en las bases de datos o a través de llamadas automatizadas con ejecución periódica. En el caso de esta aplicación se utilizan las dos últimas. En la figura 8.1, se pueden ver las funciones de la aplicación que se ejecutan en Firebase con su correspondiente activador.

8.2.1 Función automatizada *(scheduled)*

Esta función recibe el nombre de *scheduled* porque está programada para que su ejecución se haga de forma periódica sin necesidad de hacer ningún tipo de llamada a ella. Este tipo de funciones son muy útiles para realizar comprobaciones o tareas de actualización y mantenimiento de la base de datos. En el caso de *scheduled*, se puede definir como una función multiusos que se encarga de comprobar el estado de las sesiones en la base de datos y realizar acciones según estos estados. La función se ejecuta automáticamente cada minuto, y sus funciones en la aplicación son las siguientes:

- **Control de notificaciones:** cuando queden diez minutos para el comienzo de una sesión, se modifica la base de datos añadiendo el ID de esta sesión al campo notificaciones

de un evento para que se muestre el *banner* correspondiente en la vista. Un segundo antes de que empiece lo elimina.

- **Control de inicio de sesión:** cuando se elimina el *banner* y, por lo tanto, acaba el tiempo de espera, se encarga de poner el estado de la sesión en *calculatingRooms* y de llamar al algoritmo de agrupación.
- **Recopilación de datos para el algoritmo:** para que el algoritmo de agrupamiento funcione, *scheduled* se encarga de consultar los IDs de las personas que están conectadas y en sala de espera, descargar sus intereses y pasárselos al algoritmo para que este cree las diferentes salas para la sesión.
- **Escritura de datos del algoritmo:** una vez el algoritmo termina su ejecución, la función escribe los datos de las salas ya formadas en la base de datos y cambia el estado a *talking*, para que la aplicación sepa que ya pueden comenzar las reuniones por videollamada.

8.2.2 Función *voting*

Esta función se ejecuta cada vez que un usuario vota a sus compañeros de reunión. En ese caso, se añade un campo nuevo en la base de datos a tiempo real en la ruta /status/idSesion/alreadyVoted. El propósito de la función es comprobar si todos los usuarios que están conectados en una reunión han votado ya y, en caso de que esto ocurra, se llamará al algoritmo de agrupamiento de nuevo para comenzar con la siguiente ronda.

8.2.3 Función *onDisconnect*

Esta función se ejecuta cada vez que se añade o elimina un usuario en la base de datos en tiempo real, concretamente en la ruta de usuarios online. Esta función es necesaria, porque si todos los participantes menos uno han votado, y este último se desconecta, no se llamará a la función *voting* las veces necesarias y la sesión no avanzaría a la siguiente ronda de reuniones. Por esto, cada vez que alguien se desconecta, se comprueba si se puede avanzar a la siguiente ronda. Tanto en el caso de esta función como en la de *voting*, los tiempos de ejecución son ínfimos aunque tengan que acceder a la base de datos, gracias a la buena optimización de Firebase.

8.3 Funcionamiento de las sesiones

Una de las lógicas más complicadas de la aplicación móvil es quizás el funcionamiento de las denominadas sesiones. Como ya se ha comentado, el administrador de un evento puede

utilizar la aplicación web de administración para crear una serie de sesiones asociadas a los grupos de interés de ese evento en las que pueden registrarse los usuarios mediante la aplicación móvil. En esta sección, se explica paso a paso como es el flujo desde que el usuario se inscribe en una sesión hasta que esta termina y el mecanismo para habilitar las reuniones de *speed networking* en dichas sesiones.

8.3.1 Sala de espera

Una vez el usuario se ha apuntado a una sesión y quedan menos de diez minutos para su comienzo, la función *scheduled* habrá escrito el ID de esa sesión en el campo notificaciones por lo que se le mostrará el *banner* correspondiente en la aplicación. Al tocar el *banner*, se entra en una sala de espera que muestra una cuenta atrás hasta la hora de comienzo y el número de usuarios esperando. En ese momento, cuando un usuario entra en la sala de espera, es cuando se escribe el ID del usuario en la base de datos en tiempo real para que la aplicación sepa que está conectado a esa sesión.

Una vez acaba el tiempo de espera, la función *scheduled* elimina el *banner* y llama a la función de agrupar a los usuarios conectados. Una vez calculados los grupos, en la sala de espera aparece una nueva cuenta atrás de cinco segundos para avisar a los usuarios de que van a entrar en la primera sala. Al acabar la cuenta atrás, se redirige automáticamente y comienza la videoconferencia para cada grupo de usuarios.

8.3.2 Reunión

En el momento en el que un usuario entra en reunión, se hace una consulta en la base de datos para saber en qué sala se encuentra, y se envía a Agora una petición para entrar en ella. Todo esto se realiza en milésimas de segundo por lo que el usuario no lo percibe. Una vez se ha iniciado la videollamada, se muestra un contador del tiempo restante y el usuario puede empezar a hablar con sus compañeros de sala. Al finalizar el tiempo máximo establecido para la llamada, se cuelga automáticamente y la aplicación te redirige a la pantalla de votación. En caso de haber colgado antes de tiempo también te redirige a esa misma pantalla.

8.3.3 Pantalla de votación

La pantalla de votación permite que un usuario valore al resto de personas que han participado en la reunión. Contiene una cuenta atrás de forma que se permite que los usuarios tengan un minuto para realizar sus votaciones. En caso de haber colgado antes de finalizar, esta cuenta atrás será de un minuto más el tiempo que restaba de la reunión. Las imágenes y nombres de los participantes a los que se vota se cargan durante la reunión para no tener que

esperar. Una vez se acaba el tiempo o el usuario pulsa el botón de votar, se le redirige a la sala de espera intermedia si quedan más reuniones. En caso contrario, se le redirige a la sala de espera final. Es importante destacar que, si el participante no vota, se le acabará el tiempo de la cuenta atrás y se redirigirá automáticamente a la siguiente pantalla, guardando unos votos por defecto para que la aplicación sepa que debe continuar.

8.3.4 Sala de espera intermedia

Esta sala de espera es de las más cortas ya que, simplemente, se utilizar para redirigir a los usuarios que han votado rápidamente mientras esperan a que el resto acabe de votar. Una vez que todos los usuarios hayan votado, la función *voting* llama al algoritmo de agrupamiento para calcular la siguiente composición de las salas y se redirige automáticamente a los usuarios de la sesión a su siguiente llamada.

8.3.5 Sala de espera final

El procedimiento que se ha indicado hasta ahora se repite, en principio, el número de rondas estipulado para la sesión. Cuando no quedan más rondas de reuniones, la aplicación redirige a los usuarios de la sesión a la sala final. La sala de espera final tiene el mismo funcionamiento que la sala intermedia pero, como ya no hay más reuniones, realiza la redirección a una nueva sala para el cálculo de los *matches*. Es decir, cuando todo el mundo haya votado, busca en la base de datos qué personas se han votado positivamente y redirige al usuario a la pantalla de *matches*.

8.3.6 Pantalla de *matches*

La pantalla de *matches* muestra con una animación de *confetti* los nombres y fotos de los nuevos contactos que hayas realizado y un botón para finalizar la sesión. En caso de no haber conseguido ningún *match*, aparecerá un mensaje de ánimo para la próxima sesión.

8.3.7 Control de participantes

En cualquiera de las pantallas de espera, en caso de que se hayan desconectado algunos participantes y ya no haya suficientes como para generar más reuniones, los usuarios pueden ver un mensaje de que no hay suficientes personas y se les redirigirá automáticamente al final de la sesión. Este estado se representa en la base de datos como *noParticipants* y también sirve para finalizar sesiones en las que no ha participado nadie y por lo tanto no deben de comenzar.

8.4 Otras funciones a tener en cuenta

En esta sección, se explican unas cuantas funciones que no están relacionadas entre ellas pero que es interesante nombrar.

8.4.1 Traducciones

El objetivo a largo plazo de la aplicación es que esta se pueda internacionalizar. Por esa razón, para escribir todos los textos de la aplicación se han utilizado archivos JSON e, inteligentemente, la aplicación busca el idioma del teléfono e intenta utilizar el JSON correspondiente. Por el momento, sólo está disponible un JSON en español pero, para añadir más idiomas, es tan sencillo como enviar este archivo a un traductor e introducirlo en la aplicación sin modificar código. Así, un botón con el texto "guardar" no tiene escrito este texto en el código, sino que tiene una referencia a los archivos de traducciones y buscará el archivo con el idioma del teléfono. En caso de no existir, se utiliza uno por defecto, en este caso el español.

8.4.2 Temas

Es muy posible que durante la vida de la aplicación se quieran cambiar los colores de los elementos o sus fuentes y tamaños. También, cada vez más aplicaciones implementan un modo oscuro que sea menos "agresivo" para la vista de los usuarios. Por ello, todos los colores de los elementos y fuentes de la aplicación están definidos en un archivo aparte de forma que, cualquier cambio de tema, se puede realizar únicamente en este archivo y se cambiará en toda la aplicación, manteniendo la consistencia.

8.4.3 Compilación de *backend*

El *backend* se ha implementado en Dart pero, hasta ahora, no se había comentado que Firebase Functions sólo acepta JavaScript o TypeScript para sus funciones. Por ello, antes de desplegar nuestras funciones de *backend* es necesario que se "traduzcan" a código JavaScript. Este proceso se puede realizar con un sólo comando gracias a librerías de Node.js, por lo que es muy sencillo y no causa demasiado trabajo a mayores.

8.5 Organización del código

Para la parte de implementación en Flutter, se ha seguido la arquitectura *ScopedModel*, como se explicó anteriormente. Para que el código sea legible y la estructura de archivos sea limpia y comprensible, se han seguido unas pautas básicas de organización del código y de los archivos creados en Dart:

- Se creará un paquete por cada pantalla principal (inicio, contactos, sesiones, notificaciones, perfil).
- Cada paquete estará divido en tres directorios (*model*, *widgets* y *screens*).
- El directorio *model* contiene los modelos de *ScopedModel* y las clases persistentes necesarias.
- El directorio *widgets* contiene los *widgets* que se usan en las pantallas pero tienen comportamientos complejos y ocupan muchas líneas de código.
- El directorio *screens* contiene las pantallas en sí y *widgets* básicos.
- En caso de que un modelo se alargue demasiado, se crea una extensión de ese modelo. En la aplicación se ha creado un *TopicsModel* (con las funciones para configurar intereses) que extiende a *UserModel* (con los datos y funciones del usuario).
- En caso de utilizar el mismo *widget* en varias pantallas, se creará un archivo *common_widgets* para no replicar código.
- Se intentará en todo momento que, en los archivos de modelo, no haya lógica de Flutter y que, en los archivos de pantallas y *widgets*, no haya lógica de negocio.

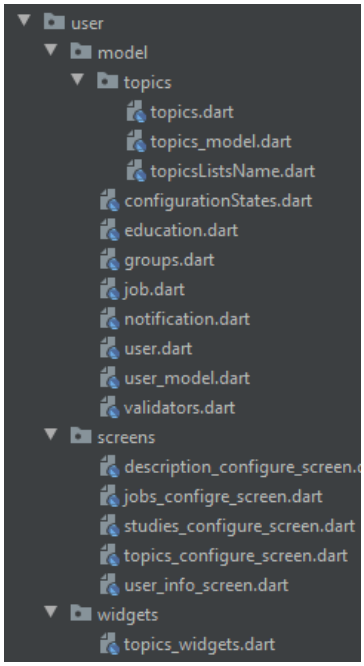

Figura 8.2: Ejemplo de organización del código.

Capítulo 9

Conclusiones y trabajo futuro

En este último capítulo de la memoria se resumirá cómo ha ido finalmente el desarrollo del proyecto y sus partes más importantes. También se expondrán los siguientes pasos a realizar para seguir mejorando la aplicación de cara a su comercialización.

9.1 Conclusiones y lecciones aprendidas

Como principal conclusión cabe destacar que se ha logrado desarrollar una aplicación con bastante complejidad técnica en un plazo de tiempo bastante corto. La aplicación resultante podrá ser, a partir del momento de la entrega, comercializable y cualquier cliente potencial podría probarla. Gracias a los conocimientos adquiridos en metodologías de programación moderna se ha logrado desarrollar la aplicación desde la aparición de la idea hasta su mínimo producto viable.

La duración del desarrollo casi no se ha visto modificada pues no ocurrieron casi retrasos durante el proceso, solamente finalizó una semana más tarde y esto teniendo en cuenta que el desarrollador trabajó solo. Esto confirma que dedicar un gran tiempo a la planificación y organización de las tareas a realizar es de gran ayuda. Aunque de primeras parezca que se está perdiendo un tiempo en el que ya se podría haber comenzado a programar, a largo plazo ha resultado de mucha utilidad y el hecho de realizar una buena planificación hizo que no surgieran nuevos casos de uso ni retrasos.

El desarrollo en Flutter ha sido muy fluido y en ningún momento se ha llegado a un punto muerto en el que no se podía avanzar puesto que, aunque se trate de un *framework* muy joven, la cantidad de documentación, foros y paquetes que ayudan al desarrollador es enorme, y seguirá creciendo conforme crezca la comunidad de usuarios de Flutter.

La aplicación web es por el momento muy sencilla como para evaluar su funcionamiento, pero la aplicación móvil se ha compilado sin problema tanto para iOS como para Android y el rendimiento es excelente. Tanto los tiempos de carga en combinación con Firebase, como la velocidad de las animaciones es muy rápida, teniendo en cuenta que la aplicación carga una gran cantidad de datos desde la base de datos y no guarda información en la memoria del dispositivo. Posiblemente, con algo de trabajo de optimización se podría acelerar aún más.

La parte más compleja de desarrollar fue el algoritmo genético para realizar las agrupaciones de usuarios aunque, una vez se realizó la correspondiente investigación, no hubo demasiados problemas. Aquí entra otra vez el tema de la planificación ya que, si se hubiera hecho una investigación de las diferentes alternativas para el algoritmo de agrupamiento antes de haber comenzado el desarrollo, posiblemente no hubiera generado un retraso y el proyecto se habría ejecutado exactamente en el tiempo planificado.

Por último, comentar que el esfuerzo de este desarrollo se ha traducido en un puesto de trabajo para continuar con el desarrollo de la idea y, posiblemente, la aplicación resultante se ponga a la venta. De esta forma, este trabajo de fin de grado también ha sido de gran ayuda para el futuro profesional del alumno, tanto desde el punto de vista de todo lo aprendido en tecnologías punteras y metodologías de trabajo como en términos de oportunidades laborales.

9.2 Trabajo futuro

Aunque se ha desarrollado un MVP, que es suficiente para probar con posibles clientes y demostrar su viabilidad, a la aplicación le queda un largo camino para ser un producto final. Los siguientes pasos a realizar, sin contar las pruebas, serían los siguientes:

- **Implementar Google Analytics:** Google Analytics es otro servicio de Firebase con librería para Flutter con el que se puede ver de forma gráfica el número de usuarios concurrentes, en qué pantalla pasan más tiempo, errores o cierres de la aplicación y todo tipo de *logs*.
- **Control de usuarios concurrentes:** Para una aplicación en producción es interesante tener un control de cuántos usuarios concurrentes participan en las reuniones para evitar sobrecargas de red y posibles caídas.
- **Login social:** uno de los objetivos a cumplir sacados del *Design Sprint* fue el de hacer un registro no muy pesado para el usuario, por lo que poder logearse con cuentas de Linkedin o Office 365 y recopilar algunos datos de ellas es otro objetivo interesante para la aplicación final.

- **Charla inicial:** sería interesante que al comienzo de cada sesión el organizador pueda dar una charla a todos los participantes para explicar cómo va a ser esta.
- **Otros tipos de reuniones:** a parte de reuniones aleatorias con un número reducido de personas, se podrían implementar otro tipo de reuniones.
- **Notificaciones *push*:** aprovechando que tenemos funciones en el *backend* que comprueban eventos cada minuto, estas funciones se podría utilizar también para enviar notificaciones al teléfono sin estar dentro de la aplicación para avisar del comienzo de un evento.
- **Chat con contactos:** por el momento sólo se guarda el nombre y trabajo de los contactos, pero es interesante que en un futuro se pueda chatear con ellos y enviar más datos de contacto si se desea.
- **Aplicación de administración:** actualmente la aplicación de administración es básica, suficiente para hacer pruebas, pero la idea es que esta sea mucho más potente y tenga control de usuarios y de invitaciones a los eventos.
- **Pantalla de inicio más completa:** la pantalla de inicio en este momento solo muestra tarjetas con los grupos de interés, pero el objetivo es que sea mucho más completa y muestre información sobre eventos, noticias, etc.

Apéndices

Apéndice A

Diseños adicionales

En este apéndice, se incluyen imágenes sobre el prototipo de la aplicación web de administrador. Estos diseños no se han incluido en el cuerpo de la memoria pues ocupan un espacio necesario para exponer otros temas de mayor importancia y no son necesarios para comprender el transcurso del proyecto.

Figura A.1: Diseño login en administrador.

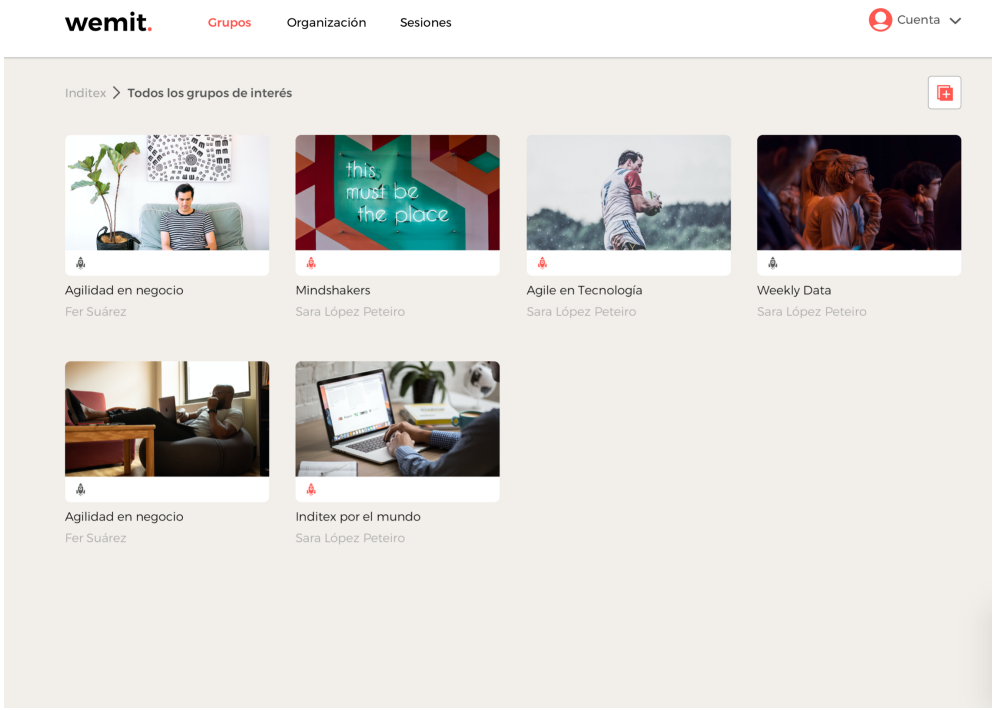

Figura A.2: Diseño de pantalla de grupos en administrador.

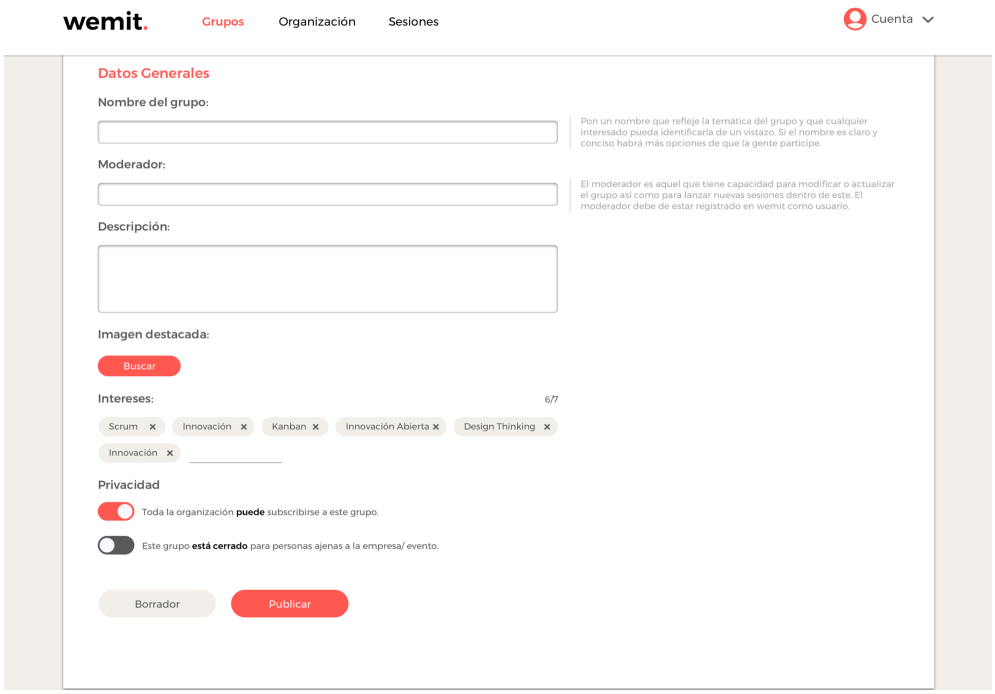

Figura A.3: Diseño de pantalla de creación de grupos en administrador.

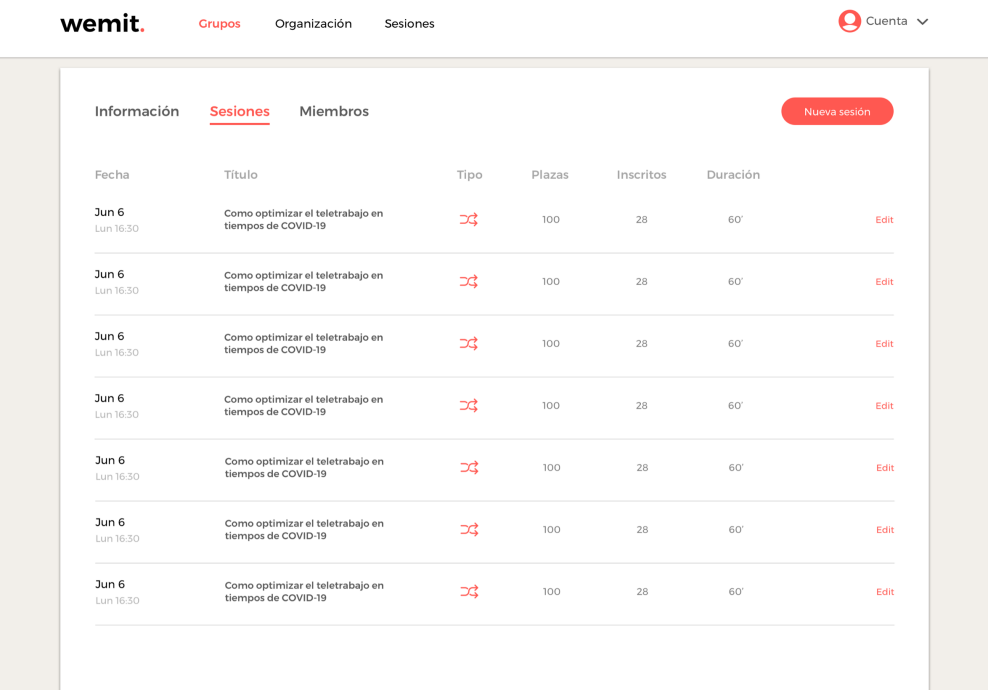

Figura A.4: Diseño de la lista de sesiones de un grupo en administrador.

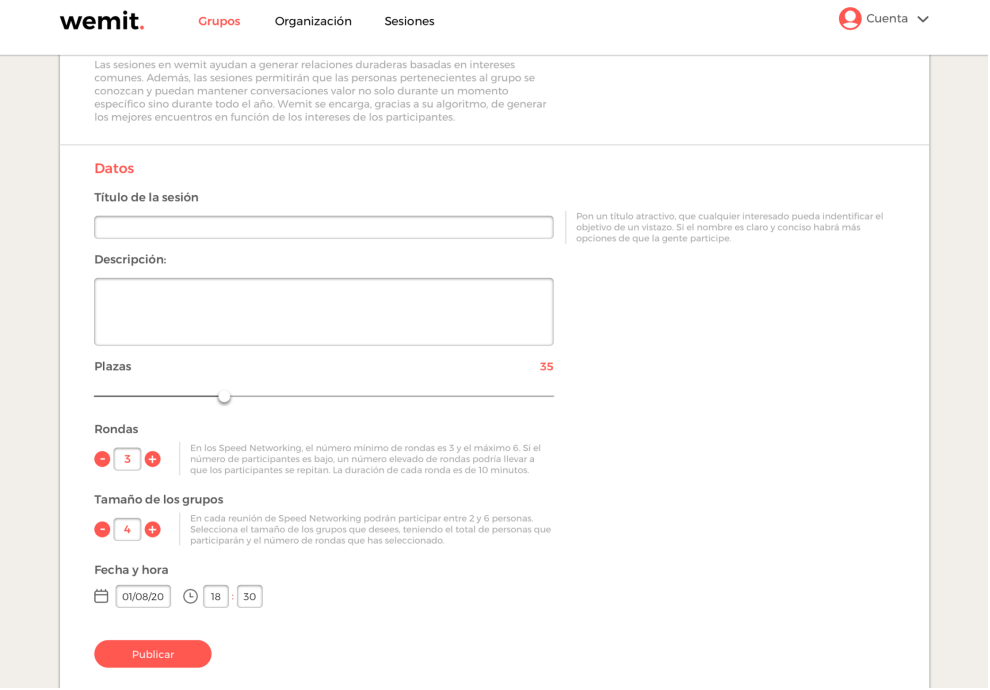

Figura A.5: Diseño para crear nueva sesión en administrador.

Apéndice B

Codemagic

En el capítulo de planificación económica se explicó que para compilar las aplicaciones para iOS es totalmente necesario disponer de un equipo con sistema operativo MacOS. Sin embargo, el coste de uno de estos dispositivos es muy alto, y puede que el desarrollador no pueda permitirse uno de estos dispositivos o decida no adquirir uno hasta el momento en el que sepa si su desarrollo tiene un futuro comercial. Por ello, en este apéndice se va a exponer un servicio que puede solucionar este problema y compilar estas aplicaciones sin necesidad de MacOS. Este servicio se llama **Codemagic** [21].

B.1 Introducción a Codemagic

Codemagic es un servicio online que permite realizar compilaciones de aplicaciones a través de máquinas virtuales con sistema operativo MacOS, ofreciendo automatizaciones tanto para compilación como para pruebas y despliegue en las tiendas de aplicaciones. Se trata de un servicio relativamente joven que apareció centrándose en Flutter, aunque permite compilar para otro tipo de *frameworks*. Según su página web, en este momento, se acercan ya al medio millón de compilaciones realizadas.

B.2 Capacidades

Entre las capacidades de la herramienta podemos destacar las siguientes:

- **Usa tu plataforma preferida:** Codemagic se centra en Flutter, pero permite compilar aplicaciones en Android, en iOS nativo o en React.
- **Remplaza procesos manuales por automáticos:** Codemagic puede compilar las aplicaciones automáticamente en cada versión.

- **Siempre actualizado:** no es necesario preocuparse por las actualizaciones de las aplicaciones necesarias para compilar pues la herramienta se encarga de ello automáticamente.
- **Test automatizados:** puedes automatizar los tests de las aplicaciones en emuladores o granjas de dispositivos reales.
- **Integración completa con Apple:** Codemagic se encarga de todos los procesos necesarios para subir la aplicación al portal de desarrolladores de Apple.
- **Publicación más rápida:** la herramienta puede desplegar automáticamente la aplicación en las distintas tiendas.
- **Integración con repositorios:** es posible integrar Codemagic con servicios basados en Git para que, automáticamente, se utilicen los últimos *commits*.
- **Elige tu máquina virtual:** puedes decidir si compilar en máquinas con más o menos potencia con costes distintos.
- **Librería de aplicaciones:** se ofrece una gran cantidad de aplicaciones preinstaladas que se pueden configurar y utilizar en el flujo de trabajo gracias a *scripts*.
- **Envío automático a testers:** Codemagic permite enviar automáticamente las compilaciones a los testers y notificarles de ello.

B.3 Precios

Codemagic ofrece un plan gratuito y otro de pago, como se muestra en la figura B.1. El plan gratuito ofrece quinientos minutos de compilaciones en máquinas Mac Mini, con una única compilación concurrente y dos puestos para trabajadores del equipo. Una vez superados estos límites, habrá que pagar 0.038 $ por cada minuto extra en las máquinas Mac Mini y 0.095 $ en las máquinas Mac Pro. Además, por cada puesto de trabajo adicional se pagarán 10$ mensuales.

B.4 Conclusiones

Como conclusión, Codemagic ofrece una buena ayuda a aquellos desarrolladores que quieran introducirse en el mundo del desarrollo para móvil y no dispongan de un ordenador Apple. Las personas que decidan usarlo tienen el plan gratuito para hacer las primeras pruebas de

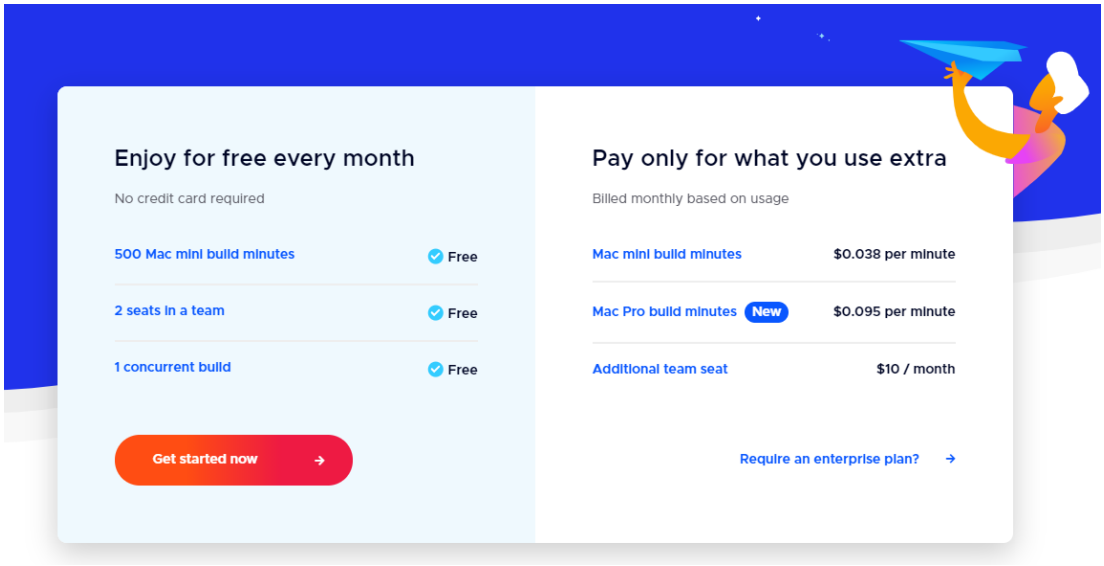

Figura B.1: Precios de la herramienta.

la aplicación, aunque necesitarán subirla a la plataforma de Apple y descargarla en un iPhone como testers antes de tenerla en producción. Para Apple siempre necesitas alguno de sus dispositivos de forma directa o indirecta pero esta herramienta facilita las cosas.

Lista de acrónimos

API Application Programming Interface. 10, 12, 38, 44, 51

HMW How Might We. v, 22

HTTP HyperText Transfer Protocol. 71

JSON JavaScript Object Notation. 53, 75

MVC Model View Controller. 61

MVP Minimal Viable Product. v, 17, 27–29, 31, 33, 40, 48, 51, 52, 64, 78

NoSQL Not Only Structured Query Language. 11, 53, 60

SMS Short Message Service. 12

URL Uniform Resource Locator. 11, 54–57, 59

Bibliografía

[1] D. Clark, "How to host a virtual networking event," 2020. [En línea]. Disponible en: https://hbr.org/2020/05/how-to-host-a-virtual-networking-event?utm_medium=email&utm_source=newsletter_daily&utm_campaign=dailyalert_activesubs&utm_content=signinnudge&referral=00563&deliveryName=DM81753

[2] "Meetic." [En línea]. Disponible en: https://www.meetic.es/

[3] "Tinder." [En línea]. Disponible en: https://tinder.com/?lang=es-ES

[4] "Linkedin." [En línea]. Disponible en: https://es.linkedin.com/

[5] "Meetup." [En línea]. Disponible en: https://www.meetup.com/es-ES/

[6] "Dart." [En línea]. Disponible en: https://dart.dev/

[7] "Flutter." [En línea]. Disponible en: https://flutter.dev/?gclid=Cj0KCQjw7sz6BRDYARIsAPHzrNKyS5cRyzHO5LYIcGdAo_mJsp939zE12oy6he0HU-mx--ymfEbFV9AaAkWBEALw_wcB&gclsrc=aw.ds

[8] "Firebase." [En línea]. Disponible en: https://firebase.google.com/

[9] "Agora." [En línea]. Disponible en: https://www.agora.io/en/

[10] "Skecth." [En línea]. Disponible en: https://www.sketch.com/

[11] "Xcode." [En línea]. Disponible en: https://developer.apple.com/xcode/

[12] "Android studio." [En línea]. Disponible en: https://developer.android.com/studio

[13] "Vscode." [En línea]. Disponible en: https://code.visualstudio.com/

[14] "Taiga." [En línea]. Disponible en: https://taiga.io/

[15] "Miro." [En línea]. Disponible en: https://miro.com/app/dashboard/

[16] "Overleaf." [En línea]. Disponible en: https://www.overleaf.com/

[17] "Gitlab." [En línea]. Disponible en: https://about.gitlab.com/

[18] "Manifiesto agile." [En línea]. Disponible en: https://agilemanifesto.org/iso/es/principles.html

[19] "Indeed." [En línea]. Disponible en: https://es.indeed.com/

[20] "Algoritmo genético." [En línea]. Disponible en: https://planetachatbot.com/entendiendo-los-algoritmos-gen%C3%A9ticos-un-caso-de-uso-en-el-entorno-organizacional-a745c157fa8c

[21] "Codemagic." [En línea]. Disponible en: https://codemagic.io/start/